S.No: 1	Exp. Name: ***Array Reverse***	Date: 2024-03-01

ID: 23K61A4752

2023-2027-CIC

Sasi Institute of Technology and Engineering (Autonomous)

Aim:

Write a C program to reverse all the elements in the array.

Input Format:

- First line of input contains an integer **N** representing the size of array
- Second line of input contains N no.of space separated integers representing the array elements

Output Format:

- Print the elements of the array in reverse order

Constraints:

- 1 <= N <= 1000
- 0 <= arr[i] <= 1000

Source Code:

ArrayReverse.c

```
#include<stdio.h>
int main()
{
        int arr[50],rev[50],s,i,j;
        scanf("%d",&s);
        for(int i=0;i<s;i++)
                {
                        scanf("%d",&arr[i]);
                }
        for(int i=s-1;i>=0;i--)
                {
                        rev[j]=arr[i];
                        j++;
                }
        for(int i=0;i<s;i++)
                {
                        printf("%d ",rev[i]);
                }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
3
15 24 62
62 24 15

Test Case - 2

User Output
4
-54 63 -21 51
51 -21 63 -54

Test Case - 3
User Output
5
-50 -60 -70 100 80
80 100 -70 -60 -50

Test Case - 4
User Output
6
12 15 19 8 63 -78
-78 63 8 19 15 12

Test Case - 5
User Output
5
-5 -10 -15 -20 -25
-25 -20 -15 -10 -5

S.No: 2	Exp. Name: ***Search for an element using Linear search***	Date: 2024-03-01

ID: 23K61A4752

Sasi Institute of Technology and Engineering (Autonomous)

Aim:

Write a C program to check whether the given element is present or not in the array of elements using linear search.

Source Code:

SearchEle.c

```
#include<stdio.h>
void main()
{
        int a[10],n,i,key,flag=0,pos;
        printf("Enter size: ");
        scanf("%d",&n);
        printf("Enter %d element: ",n);
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        printf("Enter search element: ");
        scanf("%d",&key);
        for(i=0;i<n;i++)
                {
                        if(key==a[i])
                        {
                                flag=1;
                                pos=i;
                                break;
                        }
                }
        if(flag==1)
                printf("Found at position %d\n",pos);
        else
                printf("%d is not found\n",key);

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter size:
6
Enter 6 element:
2 4 8 1 3 5
Enter search element:
6
6 is not found

Test Case - 2
User Output
Enter size:
6
Enter 6 element:
2 4 8 1 3 5
Enter search element:
2
Found at position 0

Test Case - 3
User Output
Enter size:
6
Enter 6 element:
2 4 8 1 3 5
Enter search element:
9
9 is not found

S.No: 3	Exp. Name: ***Bubble Sort***	Date: 2024-03-01

Aim:

Write a C program that reads n integer numbers and arrange them in ascending order using Bubble Sort.

Source Code:

bubbleSort.c

```
#include<stdio.h>
int main()
{
        int arr[20],n,i,j,k,temp;
        printf("n: ");
        scanf("%d",&n);
        printf("Elements: ");
        for(i=0;i<n;i++)
                {
                scanf("%d",&arr[i]);
                }
        printf("Before sorting: ");
        for(i=0;i<n;i++)
        {
                        printf("%d ",arr[i]);
                }
        for(i=0;i<n-1;i++)
                {
                        for(j=0;j<n-1;j++)
                                {
                                        if(arr[j]>arr[j+1])
                                        {
                                temp=arr[j];
                                arr[j]=arr[j+1];
                                arr[j+1]=temp;
                                }
                }
}
printf("\nAfter sorting: ");
for(i=0;i<n;i++)
        {
                printf("%d ",arr[i]);
        }
printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
n:
4
Elements:

44 22 66 11
Before sorting: 44 22 66 11
After sorting: 11 22 44 66

Test Case - 2
User Output
n:
5
Elements:
9 2 7 1 6
Before sorting: 9 2 7 1 6
After sorting: 1 2 6 7 9

S.No: 4	Exp. Name: ***Non-recursive Binary search***	Date: 2024-03-14

ID: 23K61A4752

Aim:

Write a C program that use non-recursive functions to perform the Binary search operation for a Key value in a given list of integers.

Input format:

- Size of the Array**:** An integer indicating the number of elements in the array (up to 20).
- Elements of the Array**:** A series of integers entered by the user, which should be sorted for binary search to work correctly.
- Search Element**:** An integer that the user wants to search for within the array.

Output Format:

- If the search element is found, it outputs the position of the element (1-based index).
- If the search element is not found, it outputs "not found".

Source Code:

recursiveBinarySearch.c

```
#include <stdio.h>
int main()
{
        int arr[10],num,i,n,beg,mid,end,found=0;
        printf("size: ");
        scanf("%d",&n);
        printf("elements: ");
        for(i=0;i<n;i++){
                scanf("%d",&arr[i]);
        }
        printf("search element: ");
        scanf("%d",&num);
        beg=0,end=n-1;
        while(beg<=end)
                {
                        mid=(beg+end)/2;
                        if(arr[mid]==num)
                        {
                                printf("found at %d",mid+1);
                                break;
                        }
                        else if(arr[mid]>num)
                                end=mid-1;
                        else
                                beg=mid+1;
                        if(beg>end&&found==0)
                                printf("not found");
                }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

2023-2027-CIC
Sasi Institute of Technology and Engineering (Autonomous)

User Output
size:
3
elements:
3 6 9
search element:
6
found at 2

Test Case - 2
User Output
size:
3
elements:
3 6 9
search element:
2
not found

S.No: 5	Exp. Name: ***C program that implements the Insertion sort***	**Date: 2024-03-01**

Aim:

Write a C program that implements the Insertion sort to sort a given list of integers in ascending order.

Input Format:

- The first line of the input contains an integer n representing the number of elements.
- The second line contains n space-separated integers representing the elements to be sorted.

Output Format:

- The first line will contain the array before sorting.
- The second line will contain the array after sorting using Insertion Sort.

Source Code:

insertionSort.c

```
#include<stdio.h>
int main()
{
        int a[20],i,j,n,temp,ele;
        printf("Enter no of elements: ");
        scanf("%d",&n);
        printf("Enter the elements: ");
        for(i=0;i<n;i++)
                {
                        scanf("%d",&a[i]);
                }
        printf("Array before sort: ");
        for(i=0;i<n;i++)
                {
                printf("%d ",a[i]);
                }
        for(i=0;i<n;i++)
                {
        ele=a[i];
                for(j=i;j>0;j--)
                        {
                                if(ele<a[j-1])
                                {
                                        a[j]=a[j-1];
                                        a[j-1]=ele;
                                }
                        }
                }
        printf("\nArray after insertion sort: ");
        for(i=0;i<n;i++)
                {
                        printf("%d ",a[i]);
                }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter no of elements:
6
Enter the elements:
1 5 4 2 6 8
Array before sort: 1 5 4 2 6 8
Array after insertion sort: 1 2 4 5 6 8

Test Case - 2
User Output
Enter no of elements:
8
Enter the elements:
5 2 10 36 95 14 10 23
Array before sort: 5 2 10 36 95 14 10 23
Array after insertion sort: 2 5 10 10 14 23 36 95

S.No: 6	Exp. Name: ***C program that implements the Selection sort***	Date: 2024-03-14

ID: 23K61A4752

2023-2027-CIC

Sasi Institute of Technology and Engineering (Autonomous)

Aim:

Write a C program that implements the Selection sort to sort a given list of integers in ascending order.

Source Code:

selectionSort.c

```
#include<stdio.h>
void main()
{
        int i,n,min,temp,j,a[20];
        printf("Enter no of elements: ");
        scanf("%d",&n);
        printf("Enter the elements: ");
        for(i=0;i<n;i++)
                {
                        scanf("%d",&a[i]);
                }
        printf("Array before sort: ");
        for(i=0;i<n;i++)
                printf("%d ",a[i]);
        for(i=0;i<n-1;i++)
                {
                        min=i;
                        for(j=i+1;j<n;j++)
                                {
                                        if(a[j]<a[min])
                                        {
                                                min=j;
                                        }
                                }
                        temp=a[i];
                        a[i]=a[min];
                        a[min]=temp;
                }
        printf("\nArray after sort: ");
        for(i=0;i<n;i++)
                printf("%d ",a[i]);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter no of elements:
5
Enter the elements:
2 6 1 5 7
Array before sort: 2 6 1 5 7

Array after sort: 1 2 5 6 7

Test Case - 2
User Output
Enter no of elements:
6
Enter the elements:
62 51 58 96 32 14
Array before sort: 62 51 58 96 32 14
Array after sort: 14 32 51 58 62 96

Test Case - 3
User Output
Enter no of elements:
5
Enter the elements:
64 25 12 22 11
Array before sort: 64 25 12 22 11
Array after sort: 11 12 22 25 64

S.No: 7	Exp. Name: ***Write a C program to Insert an element at End in Singly Linked List***	Date: 2024-03-22

Aim:

Write a c program to perform insertion at end and display the elements of the single linked list.

Note: Driver code is already given for you.

Source Code:

SingleLL3.c

```
#include<stdio.h>
#include<stdlib.h>

#include "InsAtEnding.c"

void main() {
        NODE first = NULL;
        int x, op;
        while(1) {
                printf("1.Insert At End 2.Traverse the List 3.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1: printf("Enter an element : ");
                                        scanf("%d", &x);
                                        first = insertAtEnd(first, x);
                                        break;
                        case 2: if (first == NULL) {
                                                printf("Single Linked List is empty\n");
                                        } else {
                                                printf("The elements in SLL are : ");
                                                traverseList(first);
                                        }
                                        break;
                        case 3: exit(0);
                }
        }
}
```

InsAtEnding.c

```
struct node{
int data;
struct node*link;
};
typedef struct node *NODE;
NODE CreateNode()
{
        NODE temp;
        temp=(NODE)malloc(sizeof(struct node));
        temp->link=NULL;
        return temp;
}
NODE insertAtEnd(NODE first,int x)
{
         NODE temp,p;
        temp=CreateNode();
        temp->data=x;
        if(first==NULL)
                first=temp;
        else
                p=first;
        while(p->link!=NULL)
                p=p->link;
        p->link=temp;
        return first;
}
void traverseList(NODE first){
        NODE p;
        p=first;
        while(p!=NULL)
                {
                        printf("%d --> ",p->data);
                        p=p->link;
                }
        printf("NULL\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
10
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
20

1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
30
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
2
The elements in SLL are : 10 --> 20 --> 30 --> NULL
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
3

Test Case - 2
User Output
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
2
Single Linked List is empty
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
99
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
29
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
59
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
2
The elements in SLL are : 99 --> 29 --> 59 --> NULL
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
3

S.No: 8	Exp. Name: ***Write a C program to Insert an element at Begin and Delete at End in Singly Linked List.***	Date: 2024-04-19

Aim:

Fill in the missing code in the below functions `insertAtBegin(NODE first, int x)` and `deleteAtEnd(NODE first)` in the file `InsAtBeginAndDelEnd.c`.

Source Code:

SingleLL2.c

```
#include<stdio.h>
#include<stdlib.h>

#include "InsAtBeginAndDelEnd.c"

void main() {
        NODE first = NULL;
        int x, op;
        while(1) {
                printf("1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1: printf("Enter an element : ");
                                        scanf("%d", &x);
                                        first = insertAtBegin(first, x);
                                        break;
                        case 2:if (first == NULL) {
                                                printf("Single Linked List is empty so
deletion is not possible\n");
                                        } else {
                                                first = deleteAtEnd(first);
                                        }
                                        break;
                        case 3: if (first == NULL) {
                                                printf("Single Linked List is empty\n");
                                        } else {
                                                printf("The elements in SLL are : ");
                                                traverseList(first);
                                        }
                                        break;
                        case 4: exit(0);
                }
        }
}
```

InsAtBeginAndDelEnd.c

```
struct node {
int data;
struct node *next;
};
typedef struct node *NODE;

NODE createNode() {
        NODE temp;
        temp=(NODE)malloc(sizeof(struct node));
        temp->next=NULL;
        return temp;
}

NODE insertAtBegin(NODE first, int x) {
        NODE temp;
        temp=createNode();
        temp->data=x;
        temp->next=first;
        return temp;
}
NODE deleteAtEnd(NODE First)
{
        NODE temp=First,t1;
        int value;
        if(First->next==NULL)
        {
                value= First->data;
                free(First);
                First=NULL;
        }
        else
        {
                while(temp->next!=NULL)
                        {
                        t1=temp;
                        temp=temp->next;
                        }
                value=temp->data;
                free(temp);
                t1->next=NULL;
        }
        printf("The deleted item from SLL : %d\n",value);
        return First;
}

void traverseList(NODE first) {
        NODE temp = first;
        while (temp != NULL){
                printf("%d --> ",temp -> data);
                temp = temp-> next;
        }
        printf("NULL\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
15
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
49
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
26
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
3
The elements in SLL are : 26 --> 49 --> 15 --> NULL
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
2
The deleted item from SLL : 15
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
3
The elements in SLL are : 26 --> 49 --> NULL
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
4

Test Case - 2
User Output
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
2
Single Linked List is empty so deletion is not possible
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
3
Single Linked List is empty
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :

1
Enter an element :
15
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
2
The deleted item from SLL : 15
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
3
Single Linked List is empty
1.Insert At Begin 2.Delete at End 3.Traverse the List 4.Exit
Enter your option :
4

S.No: 9	Exp. Name: ***Write a C program to reverse the Singly Linked List.***	Date: 2024-04-19

Aim:

Write a C program to reverse the elements of a single linked list.

Source Code:

ReverseList.c

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node*next;
};
typedef struct node *Node;
Node insertend(Node);
Node reverse(Node);
void display(Node);
void main()
{
        int i,n;
        printf("Enter no.of nodes: ");
        scanf("%d",&n);
        Node first=NULL;
        if(n<=0)
        {
                printf("List size must be greater than zero:\n");
                printf("Enter no.of nodes: ");
                scanf("%d",&n);
        }
        printf("Enter data: ");
        for(i=0;i<n;i++)
                {
                        first=insertend(first);
                }
        first=reverse(first);
        display(first);
}
Node insertend(Node first)
{
        Node temp,new_node;
        new_node=(Node)malloc(sizeof(struct node));
        scanf("%d",&new_node->data);
        if(first==NULL)
        {
                first=new_node;
                new_node->next=NULL;
        }
        else{
                temp=first;
                while(temp->next!=NULL)
                        {
                                temp=temp->next;
                        }
                temp->next=new_node;
                new_node->next=NULL;
        }
        return first;
}
Node reverse(Node first)
{
        Node cur,prev,next_node;
```

```
        while(cur!=NULL)
                {
                        next_node=cur->next;
                        cur->next=prev;
                        prev=cur;
                        cur=next_node;
                }
        first=prev;
        return first;
}
void display(Node first)
{
        printf("Reversed the list: ");
        Node temp;
        temp=first;
        while(temp!=NULL)
                {
                        printf("%d ",temp->data);
                        temp=temp->next;
                }
        printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter no.of nodes:
4
Enter data:
1 2 3 4
Reversed the list: 4 3 2 1

Test Case - 2
User Output
Enter no.of nodes:
0
List size must be greater than zero:
Enter no.of nodes:
10
Enter data:
15 12 31 14 158 140 465 235 48 49
Reversed the list: 49 48 235 465 140 158 14 31 12 15

S.No: 10	Exp. Name: ***Reverse of a single linked list recursively.***	Date: 2024-04-26

Aim:

Write a C program to reverse a single linked list recursively.

Source Code:

RecursiveReverse.c

ID: 23K61A4752

2023-2027-CIC

Sasi Institute of Technology and Engineering (Autonomous)

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};
typedef struct Node *NODE;
NODE insertend(NODE);
NODE reverse(NODE,NODE);
NODE display(NODE);
// write a function to create a new node with the given data
NODE insertend(NODE first)
{
        NODE temp,new_node;
        new_node = (NODE)malloc(sizeof(struct Node));
        scanf("%d",&new_node->data);
        if(first==NULL)
        {
                first = new_node;
                new_node->next = NULL;
        }
        else{
                temp=first;
                while(temp->next!=NULL)
                        {
                                temp=temp->next;
                        }
                temp->next=new_node;
                new_node->next=NULL;
        }
        return first;
}
NODE display(NODE first)
{
        NODE temp;
        temp=first;
        while(temp!=NULL)
                {
                        printf("%d -> ",temp->data);
                        temp=temp->next;
                }
        printf("Null\n");
        return first;
}
NODE reverse(NODE first,NODE prev)
{
        NODE cur,nextpos;
        cur = first;
        if(cur!=NULL)
        {
                nextpos = cur->next;
                cur->next=prev;
                prev=cur;
                cur = nextpos;
```

```
        else{
                first = prev;
                return first;
        }
}
void main()
{
        int i,n;
        printf("No of nodes: ");
        scanf("%d",&n);
        NODE prev=NULL,first;
        for(i=1;i<=n;i++)
                {
                        printf("Data for node %d: ",i);
                        first = insertend(first);
                }
        printf("Original linked list: ");
        display(first);
        first=reverse(first,prev);
        printf("Reversed linked list: ");
        display(first);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
No of nodes:
5
Data for node 1:
5
Data for node 2:
4
Data for node 3:
3
Data for node 4:
2
Data for node 5:
1
Original linked list: 5 -> 4 -> 3 -> 2 -> 1 -> Null
Reversed linked list: 1 -> 2 -> 3 -> 4 -> 5 -> Null

Test Case - 2
User Output
No of nodes:
7
Data for node 1:
1

Data for node 2:
2
Data for node 3:
3
Data for node 4:
4
Data for node 5:
3
Data for node 6:
2
Data for node 7:
1
Original linked list: 1 -> 2 -> 3 -> 4 -> 3 -> 2 -> 1 -> Null
Reversed linked list: 1 -> 2 -> 3 -> 4 -> 3 -> 2 -> 1 -> Null

S.No: 11	Exp. Name: ***Single Linked List operations***	Date: 2024-05-27

Aim:

Write a C program to implement a menu driven Program for the following operations on Singly Linked List (SLL)

1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display

Source Code:

sllOperations.c

ID: 23K61A4752

2023-2027-CIC

Sasi Institute of Technology and Engineering (Autonomous)

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node*next;
};
typedef struct node *NODE;
NODE insert_begin(NODE,int);
NODE insert_end(NODE,int);
NODE insert_pos(NODE,int);
NODE del_pos(NODE);
NODE del_end(NODE);
NODE del_beg(NODE);
void display(NODE);
NODE del_beg(NODE);
NODE createnode();
int main()
{
        int x,op;
        NODE first=NULL,prev=NULL;
        while(1)
                {
                        printf("1. Insert at the beginning\n2. Insert at the end\n3. Insert
at a position\n4. Delete at a position\n5. Delete from the beginning\n6. Delete from the
end\n7. Display\n8. Exit\n");
                        printf("Enter option: ");
                        scanf("%d",&op);
                        switch(op)
                                {
                                        case 1:
                                        printf("Enter the element to insert at the
beginning: ");
                                        scanf("%d",&x);
                                        first=insert_begin(first,x);
                                        break;
                                        case 2:
                                        printf("Enter the element to insert at the end: ");
                                                scanf("%d",&x);
                                        first=insert_end(first,x);
                                        break;
                                        case 3:
                                        printf("Enter the element to insert and position:
");
                                         scanf("%d",&x);
                                        first=insert_pos(first,x);
                                        break;
                                        case 4:
                                        if(first==NULL)
                                                printf("List is empty\n");
                                        else
                                                first=del_pos(first);
                                        break;
                                        case 5:
                                        first=del_beg(first);
```

```
                                        first=del_end(first);
                                        break;
                                        case 7:
                                        display(first);
                                        break;
                                        case 8:
                                        exit(0);
                                }
                }
}
NODE createnode()
{
        NODE new_node;
        new_node = (NODE)malloc(sizeof(struct node));
        new_node->next = NULL;
        return new_node;
}
NODE insert_begin(NODE first,int x)
{
        NODE new_node;
        new_node = createnode();
        new_node->data = x;
        if(first == NULL)
                first = new_node;
        else
        {
                new_node->next = first;
                first = new_node;
        }
        return first;
}
NODE insert_pos(NODE first,int x)
{
        NODE new_node,temp,prev;
        int pos,i;
        new_node=createnode();
        new_node->data = x;
        printf("Enter position: ");
        scanf("%d",&pos);
        temp = first;
        prev = first;
        if(pos<=0)
        {
                printf("Invalid position\n");
                return first;
        }
        for(i=1;i<pos;i++)
                {
                        prev = temp;
                        temp = temp->next;
                        if(temp == NULL)
                        {
                                printf("Invalid position\n");
                                return first;
                        }
```

```
        {
                new_node->next = first;
                first = new_node;
        }
        else
        {
                new_node->next = prev->next;
                prev->next = new_node;
        }
        return first;
}
NODE insert_end(NODE first,int x)
{
        NODE new_node,temp;
        new_node = createnode();
        new_node->data = x;
        if(first == NULL)
        {
                first = new_node;
        }
        else{
                temp = first;
                while(temp->next!=NULL)
                        {
                                temp = temp->next;
                        }
                temp->next = new_node;
        }
        return first;
}
NODE del_pos(NODE first)
{
        NODE temp,prev;
        int pos,i;
        printf("Enter position to delete: ");
        scanf("%d",&pos);
        temp = first;
        prev = first;
        if(pos<=0)
        {
                printf("Invalid position\n");
                return first;
        }
        for(i=1;i<pos;i++)
                {
                        prev = temp;
                        temp = temp->next;
                        if(temp==NULL)
                        {
                                printf("Invalid position\n");
                                return first;
                        }
                }
        if(pos == 1)
        {
```

```
                printf("Deleted element is %d\n",temp->data);
                free(temp);
                return first;
        }
        else{
                prev->next = temp->next;
                printf("Deleted element is %d\n",temp->data);
                free(temp);
        }
        return first;
}
NODE del_beg(NODE first)
{
        NODE temp;
        if(first == NULL)
        {
                printf("List is empty\n");
        }
        else{
                temp = first;
                first = temp->next;
                printf("Deleted element is %d\n",temp->data);
                free(temp);
        }
        return first;
}
NODE del_end(NODE first)
{
        NODE cur,prev;
        cur = first;
        if(first == NULL)
        {
                printf("List is empty\n");
        }
        else
        {
                while(cur->next != NULL)
                        {
                                prev = cur;
                                cur = cur->next;
                        }
                printf("Deleted element is %d\n",cur->data);
                free(cur);
                prev->next = NULL;
        }
        return first;
}
void display(NODE first)
{
        NODE temp;
        temp = first;
        if(first == NULL)
        {
                printf("List is empty\n");
        }
```

```
                while(temp !=NULL)
                        {
                                printf("%d -> ",temp->data);
                                temp = temp->next;
                        }
                printf("NULL\n");
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
7
List is empty
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
6
List is empty
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
5
List is empty
1. Insert at the beginning
2. Insert at the end
3. Insert at a position

```
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
4
List is empty
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
1
Enter the element to insert at the beginning:
1
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
1
Enter the element to insert at the beginning:
2
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
1
Enter the element to insert at the beginning:
3
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
```

```
8. Exit
Enter option:
7
Elements in the list: 3 -> 2 -> 1 -> NULL
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
2
Enter the element to insert at the end:
4
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
7
Elements in the list: 3 -> 2 -> 1 -> 4 -> NULL
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
3
Enter the element to insert and position:
22
Enter position:
6
Invalid position
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
```

3
Enter the element to insert and position:
22
Enter position:
2
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
7
Elements in the list: 3 -> 22 -> 2 -> 1 -> 4 -> NULL
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
4
Enter position to delete:
6
Invalid position
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
4
Enter position to delete:
2
Deleted element is 22
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit

```
Enter option:
7
Elements in the list: 3 -> 2 -> 1 -> 4 -> NULL
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
5
Deleted element is 3
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
7
Elements in the list: 2 -> 1 -> 4 -> NULL
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
6
Deleted element is 4
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
7
Elements in the list: 2 -> 1 -> NULL
1. Insert at the beginning
2. Insert at the end
3. Insert at a position
4. Delete at a position
```

5. Delete from the beginning
6. Delete from the end
7. Display
8. Exit
Enter option:
8

S.No: 12	Exp. Name: ***Write code for removing duplicate elements in Singly Linked List***	Date: 2024-04-26

ID: 23K61A4752

2023-2027-CIC

Sasi Institute of Technology and Engineering (Autonomous)

Aim:

Write a program to remove all the duplicate elements that are present in the given singly linked lists.

Sample Input and Output:

```
Enter list elements :
Enter element : 5
Enter element : 4
Enter element : 3
Enter element : 3
Enter element : 5
Enter element : 6
Enter element : -1
List before removing duplicates : 3 3 4 5 5 6
List after removing duplicates : 3 4 5 6
```

The algorithm is as follows:

```
Step-1: Take input elements of the linked list.
Step-2: Arrange the elements in sorted order.
Step-3: Traverse from the head of the sorted linked list
Step-4: While traversing, compare the current node with the next node.
Step-5: If data of the next node is the same as the current node then delete the next
node.
Step-6: Print the resultant list elements
```

Fill the missing code in the `NODE removeDuplicates` function in the file RemoveLL.c

Source Code:

SingleLL10.c

```
#include <stdio.h>
#include <stdlib.h>
#include "RemoveLL.c"

int main() {
        NODE l1;
        l1 = NULL;
        printf("Enter list elements :\n");
        l1 = createAndAddNodes(l1);
        sort(l1);
        printf("List before removing duplicates : ");
        print(l1);
        printf("\n");
        printf("List after removing duplicates : ");
        removeDuplicates(l1);
        print(l1);
}
```

RemoveLL.c

```
struct node {
        int data;
        struct node *next;
};
typedef struct node * NODE;

NODE createAndAddNodes(NODE first) {
        NODE temp, q;
        int x;
        printf("Enter element : ");
        scanf("%d", &x);
        while(x != -1) {
                temp = (NODE)malloc(sizeof(struct node));
                temp->data = x;
                temp->next = NULL;
                if(first == NULL) {
                        first = temp;
                } else {
                        q->next = temp;
                }
                q = temp;
                printf("Enter element : ");
                scanf("%d", &x);
        }
        return first;
}
void print(NODE node) {
        while (node != NULL) {
                printf("%d ", node->data);
                node = node -> next;
        }
}
NODE sort(NODE first) {
        NODE t1, t2;
        int x;
        for(t1 = first; t1 -> next != NULL; t1 = t1 -> next) {
                for(t2 = t1 -> next; t2 != NULL; t2 = t2 -> next) {
                        if (t1 -> data > t2 -> data) {
                                x = t1 -> data;
                                t1 -> data = t2 -> data;
                                t2 -> data = x;
                        }
                }
        }
        return first;
}
NODE removeDuplicates(NODE head) {
        NODE p=head;
        NODE q=head->next;
        while(q!=NULL)
                {
                        if(p->data!=q->data)
                        {
                                p=q;
                                q=q->next;
```

```
                        {
                                p->next = q->next;
                                free(q);
                                q=p->next;
                        }
                }
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter list elements :
Enter element :
5
Enter element :
4
Enter element :
3
Enter element :
3
Enter element :
5
Enter element :
6
Enter element :
-1
List before removing duplicates : 3 3 4 5 5 6
List after removing duplicates : 3 4 5 6

S.No: 13	Exp. Name: ***Polynomial Operations - Creating and Printing Polynomials using Linked Lists***	Date: 2024-05-27

Aim:

Fill in the missing code in the below program to create and print polynomial using linked lists.

Sample Input and Output:

```
Enter coeff and exp of node : 4 3
Do u want another node (y/n) : y
Enter coeff and exp of node : 5 2
Do u want another node (y/n) : y
Enter coeff and exp of node : 6 1
Do u want another node (y/n) : y
Enter coeff and exp of node : 2 0
Do u want another node (y/n) : n
The polynomial is : 4 X^ 3 ---> 5 X^ 2 ---> 6 X^ 1 ---> 2 X^ 0 ---> NULL
```

Source Code:

PolyLLMain.c

```
#include <stdio.h>
#include <stdlib.h>
#define max 20

#include "CreateAndPrintPolyLL.c"

poly create(poly head) {
        poly temp;
        char ch;
        int coeff, exp;
        do {
                temp = (poly)malloc(sizeof(struct polynomial));
                printf("Enter coeff and exp of node : ");
                scanf("%d%d", &coeff, &exp);
                temp -> coeff = coeff;
                temp -> exp = exp;
                temp -> next = NULL;
                head = addTerm(head, temp);
                printf("Do u want another node (y/n) : ");
                scanf(" %c", &ch);
        } while(ch != 'n');
        return head;
}

void main() {
        poly head = NULL;
        int ch;
        head = create(head);
        printf("The polynomial is : ");
        print(head);
}
```

CreateAndPrintPolyLL.c

```
struct polynomial {
        int coeff;
        int exp;
        struct polynomial *next;
};
typedef struct polynomial *poly;

poly addTerm(poly head, poly temp) {
poly p1,p2;
        p1=p2=head;
        if(p1==NULL){
                head=temp;
        }
        else{
                while(p1!=NULL&&p1->exp>temp->exp){
                        p2=p1;
                        p1=p1->next;
                }
                if(p1==NULL){
                        p2->next=temp;
                } else if(p1->exp==temp->exp){
                        p1->coeff=p1->coeff+temp->coeff;
                }
                else if
                        (p1->exp<temp->exp){
                                if(p2==p1){
                                        temp->next=p1;
                                        head=temp;
                                }else{
                                        temp->next=p1;
                                        p2->next=temp;
                                }
                        }
        }
        return head;
}

void print(poly head) {
        poly p1=head;
        while(p1!=NULL)
                {
                        printf("%d X^ %d ---> ",p1->coeff,p1->exp);
                        p1=p1->next;
                }
        printf("NULL\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter coeff and exp of node :

3 2
Do u want another node (y/n) :
y
Enter coeff and exp of node :
5 1
Do u want another node (y/n) :
y
Enter coeff and exp of node :
4 0
Do u want another node (y/n) :
n
The polynomial is : 3 X^ 2 ---> 5 X^ 1 ---> 4 X^ 0 ---> NULL

Test Case - 2
User Output
Enter coeff and exp of node :
4 3
Do u want another node (y/n) :
y
Enter coeff and exp of node :
5 2
Do u want another node (y/n) :
y
Enter coeff and exp of node :
3 3
Do u want another node (y/n) :
y
Enter coeff and exp of node :
2 1
Do u want another node (y/n) :
y
Enter coeff and exp of node :
7 2
Do u want another node (y/n) :
n
The polynomial is : 7 X^ 3 ---> 12 X^ 2 ---> 2 X^ 1 ---> NULL

S.No: 14	Exp. Name: ***Polynomial Operations - Adding Polynomials using Linked List***	Date: 2024-05-28

ID: 23K61A4752

2023-2027-CIC

Sasi Institute of Technology and Engineering (Autonomous)

Aim:

Write a C program to **add** two polynomials using linked lists.

Note: Driver code is provided to you in the editor.

Source Code:

PolyLLMain1.c

```
#include <stdio.h>
#include <stdlib.h>
#include "AddPolyLL.c"

poly create(poly head) {
        poly temp;
        char ch;
        int coeff, exp;
        do {
                temp = (poly)malloc(sizeof(struct polynomial));
                printf("Coeff and Power of the term: ");
                scanf("%d%d", &coeff, &exp);
                temp -> coeff = coeff;
                temp -> exp = exp;
                temp -> next = NULL;
                head = addTerm(head, temp);
                printf("Want to add more terms?(y/n): ");
                scanf(" %c", &ch);
        } while(ch != 'n');
        return head;
}

void main() {
        poly head1=NULL, head2= NULL, result = NULL;
        int ch;
        printf("First polynomial: \n");
        head1 = create(head1);
        printf("Second polynomial: \n");
        head2 = create(head2);
        result = add(head1, head2);
        printf("First polynomial: ");
        print(head1);
        printf("Second polynomial: ");
        print(head2);
        printf("Addition: ");
        print(result);
}
```

AddPolyLL.c

```
struct polynomial {
int coeff;
int exp;
struct polynomial *next;
};
typedef struct polynomial *poly;
poly addTerm(poly head,poly temp) {
        poly p1,p2;
        p1 = p2 = head;
        if(p1 == NULL) {
                head = temp;
        }
        else{
                while(p1 !=NULL && p1->exp>temp->exp) {
                        p2 = p1;
                        p1 = p1->next;
                }
                if(p1 == NULL) {
                        p2->next = temp;
                } else if (p1->exp == temp->exp) {
                        p1->coeff = p1->coeff + temp->coeff;
                } else if(p1->exp<temp->exp) {
                        if(p2 == p1) {
                                temp->next = p1;
                                head = temp;
                        } else {
                                temp->next = p1;
                                p1->next = temp;
                        }
                }
        }
        return head;
}
void print(poly head) {
        poly p1 = head;
        int k = 1;
        while(p1 != NULL)
                {
                        if(k == 1)
                        {
                                k--;
                                printf("%d X^%d",p1->coeff,p1->exp);
                        }
                        else
                                printf(" + %d X^%d",p1->coeff,p1->exp);
                        p1 = p1->next;
                }
        printf("\n");
}
poly insert (poly head,int coeff,int exp)
{
        poly temp = (poly)malloc(sizeof(struct polynomial));
        poly t1;
        t1 = head;
        temp->coeff = coeff;
```

```
                head = temp;
        else{
                while(t1->next !=NULL)
                        t1 = t1->next;
                t1->next = temp;
        }
        return head;
}
poly add(poly poly1,poly poly2) {
        poly result = NULL;
        while(poly1 !=NULL && poly2 !=NULL) {
                if(poly1->exp == poly2->exp) {
                        result = insert(result,poly1->coeff + poly2->coeff,poly1->exp);
                        poly1 = poly1->next;
                        poly2 = poly2->next;
                } else if(poly1->exp>poly2->exp) {
                        result = insert(result,poly1->coeff,poly1->exp);
                        poly1 = poly1->next;
                } else {
                        result = insert(result,poly2->coeff,poly2->exp);
                poly2 = poly2->next;
                }
        }
        while(poly1 != NULL) {
                result = insert(result,poly1->coeff,poly1->exp);
                poly1 = poly1->next;
        }
        while(poly2!= NULL) {
                result = insert(result,poly2->coeff,poly2->exp);
                poly2 = poly2->next;
        }
        return result;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
First polynomial:
Coeff and Power of the term:
2 3
Want to add more terms?(y/n):
y
Coeff and Power of the term:
4 2
Want to add more terms?(y/n):
y
Coeff and Power of the term:
6 1
Want to add more terms?(y/n):
y
Coeff and Power of the term:
8 0
Want to add more terms?(y/n):
n
Second polynomial:
Coeff and Power of the term:
1 3
Want to add more terms?(y/n):
y
Coeff and Power of the term:
3 2
Want to add more terms?(y/n):
y
Coeff and Power of the term:
5 1
Want to add more terms?(y/n):
y
Coeff and Power of the term:
7 0
Want to add more terms?(y/n):
n
First polynomial: 2 X^3 + 4 X^2 + 6 X^1 + 8 X^0
Second polynomial: 1 X^3 + 3 X^2 + 5 X^1 + 7 X^0
Addition: 3 X^3 + 7 X^2 + 11 X^1 + 15 X^0

Test Case - 2
User Output
First polynomial:
Coeff and Power of the term:
1 3

```
Want to add more terms?(y/n):
y
Coeff and Power of the term:
2 3
Want to add more terms?(y/n):
n
Second polynomial:
Coeff and Power of the term:
3 4
Want to add more terms?(y/n):
y
Coeff and Power of the term:
4 4
Want to add more terms?(y/n):
n
First polynomial: 3 X^3
Second polynomial: 7 X^4
Addition: 7 X^4 + 3 X^3
```

S.No: 15	Exp. Name: ***Implementation of double ended queue using linked list***	Date: 2024-05-28

Aim:

Implementation of double ended queue using linked listto perform the following operations
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit

Source Code:

dooublyLinkedList.c

ID: 23K61A4752
2023-2027-CIC
Sasi Institute of Technology and Engineering (Autonomous)

```
#include<stdio.h>
#include<stdlib.h>
struct node{
int data;
struct node *next;
};
typedef struct node *NODE;
NODE rear = NULL, front = NULL;
NODE createNode() {
        NODE temp;
        temp = (NODE)malloc(sizeof(struct node));
        temp->next = NULL;
        return temp;
}
void insertRear(int x) {
        NODE temp, p;
        temp = createNode();
        temp->data = x;
        if(rear == NULL)
                rear = front = temp;
        else {
                rear->next = temp;
                rear = temp;
        }
}
void insertFront(int x) {
        NODE temp, p;
        temp = createNode();
        temp->data = x;
        if(front == NULL)
                rear = front = temp;
        else {
                temp->next = front;
                front = temp;
        }
}
void deleteFront() {
        NODE temp = front;
        if(front == rear)
                front = rear = NULL;
        else
                front = front->next;
        printf("The deleted element from Front : %d\n", temp->data);
        free(temp);
}
void deleteRear() {
        NODE temp = rear;
        if(front == rear)
                front = rear = NULL;
        else {
                NODE p1;
                p1 = front;
                while(p1->next !=rear)
                        p1 = p1->next;
                rear=p1;
```

```
        printf("The deleted element from Rear : %d\n",temp->data);
        free(temp);
}
void print() {
        NODE temp = front;
        while(temp){
                printf("%d->",temp->data);
                temp = temp->next;
        }
        printf("NULL\n");
}
void main() {
        int data, choice;
        while(1){
                printf("1.Insert at Front\n2.Insert at Rear\n3.Delete from Front\n4.Delete
from Rear\n5.Display\n6.Exit\n");
                printf("Enter your choice:\n");
                scanf("%d",&choice);
                switch (choice) {
                        case 1:
                        printf("Enter an element to Insert at Front:");
                        scanf("%d", &data);
                        insertFront(data);
                        break;
                        case 2:
                        printf("Enter an element to Insert at Rear:");
                        scanf("%d", &data);
                        insertRear(data);
                        break;
                        case 3:
                        if(front == NULL)
                                printf("Deque is empty\n");
                        else
                                deleteFront();
                        break;
                        case 4:
                        if(rear == NULL)
                                printf("Deque is empty\n");
                        else
                                deleteRear();

                        break;
                        case 5:
                        if(front == NULL)
                                printf("Deque is empty\n");
                        else
                                print();
                        break;
                        case 6:
                        exit(0);
                        default:
                        printf("Enter a valid choice");
                        break;
                }
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit
Enter your choice:
5
Deque is empty
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit
Enter your choice:
3
Deque is empty
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit
Enter your choice:
4
Deque is empty
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit
Enter your choice:
1
Enter an element to Insert at Front:
3
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit
Enter your choice:
5
3->NULL

```
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit
Enter your choice:
2
Enter an element to Insert at Rear:
5
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit
Enter your choice:
5
3->5->NULL
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit
Enter your choice:
3
The deleted element from Front : 3
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit
Enter your choice:
5
5->NULL
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit
Enter your choice:
4
The deleted element from Rear : 5
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
```

6.Exit
Enter your choice:
5
Deque is empty
1.Insert at Front
2.Insert at Rear
3.Delete from Front
4.Delete from Rear
5.Display
6.Exit
Enter your choice:
6

S.No: 16	Exp. Name: ***Write a Program to Insert an element at End and Traverse the Nodes in Doubly Linked List***	Date: 2024-05-28

Aim:

Fill in the missing code in the `insertAtEndInDLL(NODE first, int x)` and `traverseListInDLL(NODE first)` methods.

The `insertAtEndInDLL()` function adds an element to the end of the list.

The `traverseListInDLL()` function traverses and prints all the elements of the list.

Source Code:

DoubleLL1.c

```
#include<stdio.h>
#include<stdlib.h>

#include "InsertEndAndTraverseInDLL.c"

void main() {
        NODE first = NULL;
        int x, op;
        while(1) {
                printf("1.Insert At End 2.Traverse the List 3.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1: printf("Enter an element : ");
                                scanf("%d", &x);
                                first = insertAtEndInDLL(first, x);
                                break;
                        case 2: if (first == NULL) {
                                        printf("Double Linked List is empty\n");
                                } else {
                                        printf("The elements in DLL are : ");
                                        traverseListInDLL(first);
                                }
                                break;
                        case 3: exit(0);
                }
        }
}
```

InsertEndAndTraverseInDLL.c

```
struct node {
        int data;
        struct node *prev;
        struct node *next;
};
typedef struct node * NODE;

NODE createNodeInDLL() {
        NODE temp;
        temp = (NODE)malloc(sizeof(struct node));
        temp->prev = NULL;
        temp->next = NULL;
        return temp;
}

NODE insertAtEndInDLL(NODE first, int x) {
        NODE newnode, temp;
        newnode = createNodeInDLL();
        newnode->data = x;
        if(first == NULL)
                first = newnode;
        else
        {
                temp = first;
                while(temp->next !=NULL)
                        temp = temp->next;
                temp->next = newnode;
                newnode->prev = temp;
        }
        return first;
}

void traverseListInDLL(NODE first) {
        NODE temp;
        temp = first;
        while(temp != NULL) {
                printf("%d <--> ", temp->data);
                temp = temp->next;
        }
        printf("NULL\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
14
1.Insert At End 2.Traverse the List 3.Exit

Enter your option :
1
Enter an element :
67
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
56
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
2
The elements in DLL are : 14 <--> 67 <--> 56 <--> NULL
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
1
Enter an element :
34
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
2
The elements in DLL are : 14 <--> 67 <--> 56 <--> 34 <--> NULL
1.Insert At End 2.Traverse the List 3.Exit
Enter your option :
3

S.No: 17	Exp. Name: ***Implement double linked list***	Date: 2024-05-28

Aim:

Write a C program to implement double linked list and its operations

Source Code:

AllOperationsDLL.c

ID: 23K61A4752

Sasi Institute of Technology and Engineering (Autonomous)

```
#include<stdio.h>
#include<stdlib.h>
struct node {
int data;
struct node *next, *prev;
};
typedef struct node *NODE;
NODE createNode() {
        NODE temp;
        temp = (NODE)malloc(sizeof(struct node));
        temp->next = NULL;
        temp->prev = NULL;
                return temp;
}
NODE insertAtBegin(NODE first, int x) {
        NODE temp;
        temp = createNode();
        temp->data = x;
        temp->next = first;
        if(first != NULL)
                first->prev = temp;
        return temp;
}
NODE deleteAtBegin(NODE first) {
        NODE temp = first;
        int value;
        value = first->data;
        if(first->next == NULL)
        {
                free(first);
                first = NULL;
        }
        else
        {
                first = first->next;
                first->prev = NULL;
                free(temp);
        }
        printf("The deleted element from DLL : %d\n",value);
        return first;
}
void search(NODE first, int X) {
        NODE temp = first;
        int pos = 0;
        while(temp != NULL) {
                pos ++;
                if(temp->data == X)
                        break;
                temp = temp->next;
        }
        if(temp == NULL)
                printf("The given element %d is not found in the given DLL\n", X);
        else
                printf("The given element %d is found at position : %d\n", X, pos);
}
```

```
        while(temp != NULL) {
                printf("%d <--> ", temp->data);
                temp = temp->next;
        }
        printf("NULL\n");
}
void main() {
        NODE first = NULL;
        int X, op;
        while(1) {
                printf("1.Insert At Begin\n2.Delete at Begin\n3.Search an element
Position\n4.Traverse the List\n5.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1:
                        printf("Enter an element: ");
                        scanf("%d", &X);
                        first = insertAtBegin(first,X);
                        break;
                        case 2:
                        if(first == NULL) {
                                printf("Double Linked List is empty so deletion is not
possible\n");
                        } else{
                                first = deleteAtBegin(first);
                        }
                        break;
                        case 3:
                        printf("Enter search element: ");
                        scanf("%d",&X);
                        search(first,X);
                        break;
                        case 4:
                        if(first == NULL){
                                printf("Double Linked List is empty\n");
                        } else {
                                printf("The elements in DLL are: ");
                                traverseList(first);
                        }
                        break;
                        case 5:
                        exit(0);
                }
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
1
Enter an element:
15
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
2
The deleted element from DLL : 15
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
1
Enter an element:
12
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
1
Enter an element:
16
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
1
Enter an element:
17
1.Insert At Begin

2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
4
The elements in DLL are: 17 <--> 16 <--> 12 <--> NULL
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
3
Enter search element:
16
The given element 16 is found at position : 2
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
4
The elements in DLL are: 17 <--> 16 <--> 12 <--> NULL
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
5

Test Case - 2
User Output
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
2
Double Linked List is empty so deletion is not possible
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :

4
Double Linked List is empty
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
1
Enter an element:
101
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
1
Enter an element:
102
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
1
Enter an element:
103
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
6
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
2
The deleted element from DLL : 103
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :

4
The elements in DLL are: 102 <--> 101 <--> NULL
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
3
Enter search element:
103
The given element 103 is not found in the given DLL
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
4
The elements in DLL are: 102 <--> 101 <--> NULL
1.Insert At Begin
2.Delete at Begin
3.Search an element Position
4.Traverse the List
5.Exit
Enter your option :
5

S.No: 18	Exp. Name: ***Double Linked List Operations***	Date: 2024-05-28

Aim:

Write a C program to implement a menu-driven program for the following operations on Doubly Linked List (DLL) of Employee Data with the fields:

SSN, Name, Dept, Designation, Salary, PhNo

8. Create a DLL of N Employees Data by using end insertion.
9. Display the status of DLL and count the number of nodes in it
10. Perform Insertion and Deletion at End of DLL
11. Perform Insertion and Deletion at Front of DLL
12. Exit

Source Code:

`dllOps.c`

ID: 23K61A4752

2023-2027-CIC

Sasi Institute of Technology and Engineering (Autonomous)

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct node{
char ssn[25],name[25],dept[50],designation[25];
int sal;
long long int phone;
struct node *llink;
struct node *rlink;
};
typedef struct node *NODE;
NODE first = NULL;
int count=0;
NODE create()
{
        NODE enode;
        enode = (NODE)malloc(sizeof(struct node));
        if(enode == NULL)
        {
                printf("\nRunning out of memory");
                exit(0);
        }
        printf("Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee: ");
        scanf("%s %s %s %s %d %lld",enode->ssn,enode->name,enode->dept,enode-
>designation,&enode->sal,&enode->phone);
        enode->llink=NULL;
        enode->rlink=NULL;
        count++;
        return enode;
}
NODE insertfront(){
        NODE temp;
        temp=create();
        if(first==NULL)
                return temp;
        temp->rlink=first;
        first->llink=temp;
        return temp;
}
void display()
{
        NODE cur;
        cur=first;
        if(cur==NULL)
                printf("DLL is Empty\n");
        else{
                while(cur!=NULL)
                        {
                                printf("SSN:%s| Name:%s| Department:%s| Designation:%s|
Salary:%d| Phone no:%lld",cur->ssn,cur->name,cur->dept,cur->designation,cur->sal,cur-
>phone);
                                cur=cur->rlink;
                                printf("\n");
                        }
                printf("No of employees: %d\n",count);
```

```
NODE deletefront()
{
NODE temp;
if(first==NULL)
{
        printf("DLL is empty\n");
        return NULL;
}
if(first->rlink==NULL)
{
        printf("employee with ssn: %s is deleted\n",first->ssn);
        free(first);
        count--;
        return NULL;
}
temp=first;
first=first->rlink;
temp->rlink=NULL;
first->llink=NULL;
printf("employee with ssn: %s is deleted\n",temp->ssn);
free(temp);
count--;
return first;
}
NODE insertend()
{
        NODE cur,temp;
        temp=create();
        if(first==NULL)
        {
                return temp;
        }
        cur=first;
        while(cur->rlink!=NULL){
                cur=cur->rlink;
        }
        cur->rlink=temp;
        temp->llink=cur;
        return first;
}
NODE deleteend()
{
        NODE prev,cur;
        if(first==NULL)
        {
                printf("DLL is empty\n");
                return NULL;
        }
        if(first->rlink==NULL)
        {
                printf("employee with ssn: %s is deleted\n",first->ssn);
                free(first);
                count--;
                return NULL;
        }
```

```
        while(cur->rlink!=NULL)
                {
                        prev=cur;
                        cur=cur->rlink;
                }
        cur->rlink=NULL;
        printf("employee with ssn: %s is deleted\n",cur->ssn);
        free(cur);
        prev->rlink=NULL;
        count--;
        return first;
}
void main()
{
        int ch,i,n;
        while(1){
                printf("1: Create DLL of Employee Nodes");
        printf("\n2: DisplayStatus");
        printf("\n3: InsertAtEnd");
        printf("\n4: DeleteAtEnd");
        printf("\n5: InsertAtFront");
        printf("\n6: DeleteAtFront");
        printf("\n7: Exit");
        printf("\nPlease enter your choice: ");
        scanf("%d",&ch);
        switch(ch)
                {
                        case 1:printf("Enter no of Employees: ");
                        scanf("%d",&n);
                        for(i=1;i<=n;i++)
                                first=insertend();
                        break;
                        case 2:display();
                        break;
                        case 3:first=insertend();
                        break;
                        case 4:first=deleteend();
                        break;
                        case 5:first=insertfront();
                        break;
                        case 6:first=deletefront();
                        break;
                        case 7:exit(0);
                        default :printf("Please Enter valid choice\n");
                }
        }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1: Create DLL of Employee Nodes

2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
DLL is Empty
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
1
Enter no of Employees:
2
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
CT156 Hema Support PSE 30000 1234567890
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
CT188 Prasanth Support PSE 30000 1234567890
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
SSN:CT188| Name:Prasanth| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
No of employees: 2
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
3
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
CT226 Swathi Support PSE 30000 1234567890
1: Create DLL of Employee Nodes

2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
SSN:CT188| Name:Prasanth| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
SSN:CT226| Name:Swathi| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
No of employees: 3
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
4
employee with ssn: CT226 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
SSN:CT188| Name:Prasanth| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
No of employees: 2
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
5
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
CT156 Bhanu Support PSE 34000 1234567890
1: Create DLL of Employee Nodes

2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT156| Name:Bhanu| Department:Support| Designation:PSE| Salary:34000| Phone no:1234567890
SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
SSN:CT188| Name:Prasanth| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
No of employees: 3
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT156| Name:Bhanu| Department:Support| Designation:PSE| Salary:34000| Phone no:1234567890
SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
SSN:CT188| Name:Prasanth| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
No of employees: 3
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
6
employee with ssn: CT156 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
4
employee with ssn: CT188 is deleted
1: Create DLL of Employee Nodes

2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT156| Name:Hema| Department:Support| Designation:PSE| Salary:30000| Phone no:1234567890
No of employees: 1
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
6
employee with ssn: CT156 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
DLL is Empty
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
4
DLL is empty
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
6
DLL is empty

1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
3
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
198 Tanjiro Anime Hero 49000 1029384756
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:198| Name:Tanjiro| Department:Anime| Designation:Hero| Salary:49000| Phone no:1029384756
No of employees: 1
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
4
employee with ssn: 198 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
7

Test Case - 2
User Output
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd

5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
3
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
CT590 Japan Korea Indonesia 39000 0987654321
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:CT590| Name:Japan| Department:Korea| Designation:Indonesia| Salary:39000| Phone no:987654321
No of employees: 1
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
6
employee with ssn: CT590 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
DLL is Empty
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
5
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:

AP996 Sampath Support PSE 15000 9086745231
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:AP996| Name:Sampath| Department:Support| Designation:PSE| Salary:15000| Phone no:9086745231
No of employees: 1
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
4
employee with ssn: AP996 is deleted
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
DLL is Empty
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
1
Enter no of Employees:
1
Enter ssn, Name, Department, Designation, Salary, PhoneNo of employee:
TS289 Kalkanrat PublicSector Government 58000 1029238845
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd

4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
2
SSN:TS289| Name:Kalkanrat| Department:PublicSector| Designation:Government| Salary:58000| Phone no:1029238845
No of employees: 1
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
8
Please Enter valid choice
1: Create DLL of Employee Nodes
2: DisplayStatus
3: InsertAtEnd
4: DeleteAtEnd
5: InsertAtFront
6: DeleteAtFront
7: Exit
Please enter your choice:
7

S.No: 19	Exp. Name: ***Write Code for insertAtBeginInCLL() and countInCLL() functions in CLL***	Date: 2024-05-28

ID: 23K61A4752

2023-2027-CIC

Sasi Institute of Technology and Engineering (Autonomous)

Aim:

Fill in the missing code in the below functions `insertAtBeginInCLL(NODE first, int x)` and `countInCLL(NODE first)` in the file `InsAtBeginAndCountInCLL.c`.

The `insertAtBeginInCLL(NODE first, int x)` function inserts a new node at the beginning of the circular linked list.

The `countInCLL(NODE first)` function counts the number of nodes linked in a circular linked list.

Source Code:

CircularLL2.c

```
#include <stdio.h>
#include <stdlib.h>

#include "InsAtBeginAndCountInCLL.c"

void main() {
        NODE first = NULL;
        int x, op;
        while(1) {
                printf("1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List
4.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1: printf("Enter an element : ");
                                        scanf("%d", &x);
                                        first = insertAtBeginInCLL(first, x);
                                        break;
                        case 2: printf("The number of nodes in a CLL are : %d\n",
countInCLL(first));
                                        break;
                        case 3: if (first == NULL) {
                                                printf("Circular Linked List is empty\n");
                                        } else {
                                                printf("The elements in CLL are : ");
                                                traverseListInCLL(first);
                                        }
                                        break;
                        case 4: exit(0);
                }
        }
}
```

InsAtBeginAndCountInCLL.c

```
struct node {
        int data;
        struct node *next;
};
typedef struct node *NODE;

NODE createNodeInCLL() {
        NODE temp;
        temp = (NODE) malloc(sizeof(struct node));
        temp -> next = NULL;
        return temp;
}

NODE insertAtBeginInCLL(NODE first, int x) {
        NODE newnode=createNodeInCLL();
        NODE ptr=first;
        newnode->data=x;
        if(first==NULL)
        {
                first=newnode;
                newnode->next=newnode;
        }
        else
        {
                do{
                        ptr=ptr->next;
                }while(ptr->next!=first);
                newnode->next=first;
                first=newnode;
                ptr->next=newnode;
        }
        return first;
}

int countInCLL(NODE first) {
        int cnt=1;
        NODE ptr=first;
        if(first==NULL)
                return 0;
        while(ptr->next!=first){
                ptr=ptr->next;
                cnt++;
        }
        return cnt;
}

void traverseListInCLL(NODE first) {
        NODE temp = first;
        do {
                printf("%d --> ", temp -> data);
                temp = temp -> next;
        } while (temp != first);
        printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
11
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
22
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
2
The number of nodes in a CLL are : 2
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
3
The elements in CLL are : 22 --> 11 -->
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
33
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
44
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
3
The elements in CLL are : 44 --> 33 --> 22 --> 11 -->
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
2
The number of nodes in a CLL are : 4
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
4

Test Case - 2
User Output
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit

Enter your option :
3
Circular Linked List is empty
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
2
The number of nodes in a CLL are : 0
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
1
Enter an element :
99
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
2
The number of nodes in a CLL are : 1
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
3
The elements in CLL are : 99 -->
1.Insert At Begin 2.Count Number of Nodes 3.Traverse the List 4.Exit
Enter your option :
4

S.No: 20	Exp. Name: ***C program which performs all operations in Circular linked list.***	Date: 2024-05-31

Aim:

Write a program that uses functions to perform the following operations on circularlinked list.
i) Creation
ii) Insertion
iii) Deletion
iv) Traversal

Source Code:

AlloperationsinCLL.c

ID: 23K61A4752

Sasi Institute of Technology and Engineering (Autonomous)

```
#include<stdio.h>
#include<stdlib.h>
struct node {
int data;
struct node *next;
};
typedef struct node *NODE;

NODE createNodeInCLL() {
        NODE temp;
        temp=(NODE)malloc(sizeof(struct node));
        return temp;
}
NODE insertAtBeginInCLL(NODE first,int x) {
        NODE newnode=createNodeInCLL();
        NODE ptr=first;
        newnode->data=x;
        if(first==NULL)
        {
                first=newnode;
                newnode->next=newnode;
        }
        else
        {
                do{
                        ptr=ptr->next;
                } while(ptr->next!=first);;
                newnode->next=first;
                first=newnode;
                ptr->next=newnode;
        }
        return first;
}
NODE deleteFromBeginInCLL(NODE first) {
        NODE ptr=first;
        if(ptr->next==first)
        {
                printf("The deleted element from CLL : %d\n",ptr->data);
                free(ptr);
                first=NULL;
        }
        else
        {
                do{
                        ptr=ptr->next;
                } while(ptr->next!=first);
                ptr->next=first->next;
                printf("The deleted element from CLL : %d\n",first->data);
                free(first);
                first=ptr->next;
        }
        return first;
}
NODE deleteFromEndInCLL(NODE first) {
        NODE ptr=first,preptr;
```

```
                printf("The deleted element from CLL : %d\n",preptr->data);
                free(ptr);
                first=NULL;
        }
        else
        {
                do{
                        preptr=ptr;
                        ptr=ptr->next;
                        } while(ptr->next!=first);
                        preptr->next=ptr->next;
                printf("The deleted element from CLL : %d\n",ptr->data);
                free(ptr);
        }
        return first;
}
NODE insertAtEndInCLL(NODE first,int x) {
        NODE newnode=createNodeInCLL();
        NODE ptr=first;
        newnode->data=x;
        if(first==NULL)
        {
                first=newnode;
                newnode->next=newnode;
        }
        else
        {
do{
        ptr=ptr->next;
} while(ptr->next!=first);
        newnode->next=first;
        ptr->next=newnode;
}
return first;
}
int countInCLL(NODE first) {
        int cnt=1;
        NODE ptr=first;
        if(first==NULL)
                return 0;
        while(ptr->next!=first) {
                ptr=ptr->next;
                cnt++;
        }
        return cnt;
}
NODE insertAtPos(NODE first,int x,int pos)
{
        if(pos==1)
                first=insertAtBeginInCLL(first,x);
        else if(pos==countInCLL(first)+1)
                first=insertAtEndInCLL(first,x);
        else
        {
                NODE newnode=createNodeInCLL();
```

```
                for(int i=1;i<pos-1;i++)
                        preptr=preptr->next;
                newnode->next=preptr->next;
                preptr->next=newnode;
        }
        return first;
}
NODE deleteFromPos(NODE first,int pos)
{
        if(pos==1)
                first=deleteFromBeginInCLL(first);
        else if(pos==countInCLL(first))
                first=deleteFromEndInCLL(first);
        else
        {
                NODE preptr=first,pre;
                for(int i=1;i<pos;i++)
                        {
                                pre=preptr;
                                preptr=preptr->next;
                        }
                printf("The deleted element from CLL : %d\n",preptr->data);
                pre->next=preptr->next;
                free(preptr);
        }
        return first;
}
void traverseListInCLL(NODE first) {
        NODE temp=first;
        do{
                printf("%d --> ",temp->data);
                temp=temp->next;
        } while(temp!=first);
        printf("\n");
}
void main() {
        NODE first=NULL;
        int x,op,pos;
        while(1) {
                printf("1.Insert 2.Delete 3.Print 4.Exit\n");
                printf("Enter your option: ");
                scanf("%d",&op);
                switch(op) {
                        case 3: if(first==NULL)
                                printf("Circular Linked List is empty\n");
                        else {
                                printf("The elements in CLL are: ");
                                traverseListInCLL(first);
                        }
                        break;
                        case 1: printf("Enter a position: ");
                        scanf("%d",&pos);
                        printf("Enter an element: ");
                        scanf("%d", &x);
                        if(pos>0 &&pos<=countInCLL(first)+1)
```

```
                                printf("No such position in CLL so insertion is not
possible\n");
                        break;
                        case 2:if(first==NULL)
                                printf("Circular Linked List is empty so deletion is not
possible\n");
                        else{
                                printf("Enter position : ");

                                scanf("%d",&pos);
                                if(pos>0 &&pos<=countInCLL(first))
                                        first=deleteFromPos(first,pos);
                                else
                                        printf("No such position in CLL so deletion is not
possible\n");
                        }
                        break;
                        case 4: exit(0);
                        }
                }
        }
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
1
Enter an element:
1
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
2
Enter an element:
2
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
3
Enter an element:
3
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
4
Enter an element:
4
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
5
Enter an element:
5
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
6
Enter an element:
6
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:

```
3
The elements in CLL are: 1 --> 2 --> 3 --> 4 --> 5 --> 6 -->
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
2
Enter position :
3
The deleted element from CLL : 3
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
2
Enter position :
3
The deleted element from CLL : 4
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
2
Enter position :
3
The deleted element from CLL : 5
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
3
The elements in CLL are: 1 --> 2 --> 6 -->
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
3
Enter an element:
3
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
4
Enter an element:
4
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
5
Enter an element:
5
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
3
The elements in CLL are: 1 --> 2 --> 3 --> 4 --> 5 --> 6 -->
```

1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
4

Test Case - 2
User Output
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
2
Circular Linked List is empty so deletion is not possible
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
3
Circular Linked List is empty
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
1
Enter an element:
15
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
1
Enter a position:
3
Enter an element:
17
No such position in CLL so insertion is not possible
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
2
Enter position :
3
No such position in CLL so deletion is not possible
1.Insert 2.Delete 3.Print 4.Exit
Enter your option:
4

S.No: 21	Exp. Name: ***Stack using Arrays***	Date: 2024-05-31

Aim:

Write a C program to implement stack operations using arrays.

Input Format

The program presents a menu with six options. The user inputs a choice corresponding to one of these options:

13. **Push Operation:** Input is an integer value to push onto the stack.
14. **Pop Operation:** No additional input is required.
15. **Display Operation:** No additional input is required.
16. **Is Empty Operation:** No additional input is required.
17. **Peek Operation:** No additional input is required.
18. **Exit Operation:** No additional input is required.

Output Format

The output will vary based on the selected option:

19. **Push Operation:**
    iv. If the stack is not full, the output will be: **Successfully pushed**
    iv. If the stack is full, the output will be: **Stack is overflow**
22. **Pop Operation:**
    iv. If the stack is not empty, it will print: **Popped value: X** where X is the element removed from the stack.
    iv. If the stack is empty, it will print: **Stack is underflow**
25. **Display Operation:**
    iv. If the stack is not empty, it will print: **Elements: X Y Z ...** where X, Y, Z, etc., are the elements of the stack from top to bottom.
    iv. If the stack is empty, it will print: **Stack is empty**
28. **Is Empty Operation:**
    iv. If the stack is empty, it will print: **Stack is empty**
    iv. If the stack is not empty, it will print: **Stack is not empty**
31. **Peek Operation:**
    iv. If the stack is not empty, it will print: **Peek value: X** where X is the top element of the stack.
    iv. If the stack is empty, it will print: **Stack is underflow**
34. **Exit Operation:**
    iv. The program will terminate with no additional output beyond the program's exit.

Source Code:

StackUsingArray.c

```c
#include <stdio.h>
#include <stdlib.h>
#define STACK_MAX_SIZE 10
#include "StackOperations.c"

int main() {
        int op, x;
        while(1) {
                printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit\n");
                printf("Option: ");
                scanf("%d", &op);
                switch(op) {
                        case 1:
                                printf("element: ");
                                scanf("%d", &x);
                                push(x);
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                peek();
                                break;
                        case 6:
                                exit(0);
                }
        }
}
```

StackOperations.c

```
int stack[STACK_MAX_SIZE];
int top=-1;

void push(int element)  {
        if(top==STACK_MAX_SIZE-1)
                printf("Stack is overflow\n");
        else{
                stack[++top]=element;
                printf("Successfully pushed\n");
        }
}

void display() {
        if(top==-1)
                printf("Stack is empty\n");
        else{
                printf("Elements: ");
                for(int i=top;i>=0;--i)
                        printf("%d ",stack[i]);
                printf("\n");
        }
}

void pop() {
        if(top==-1)
                printf("Stack is underflow\n");
        else{
                printf("Popped value: %d\n",stack[top--]);
        }
}
void peek() {
        if(top==-1)
                printf("Stack is underflow\n");
        else
                printf("Peek value: %d\n",stack[top]);
}
void isEmpty() {
        if(top==-1)
                printf("Stack is empty\n");
        else
                printf("Stack is not empty\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
4
Stack is empty
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Option:
2
Stack is underflow
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
3
Stack is empty
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
5
Stack is underflow
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
25
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
26
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
3
Elements: 26 25
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
2
Popped value: 26
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
4
Stack is not empty
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
5
Peek value: 25
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
6

Test Case - 2
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1

```
element:
1
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
2
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
3
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
4
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
5
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
6
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
7
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
8
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
```

9
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
10
Successfully pushed
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
1
element:
11
Stack is overflow
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Option:
6

S.No: 22	Exp. Name: ***C program to implement different Operations on Stack using Linked Lists***	**Date: 2024-05-31**

Aim:

Write a program to implement **stack** using **linked lists**.

Input Format

The user is presented with a menu of options and provides input according to the desired operation:

36. **Push Operation:**
    nt. Input: Integer value to be pushed onto the stack.
38. **Pop Operation:**
    nt. No additional input is required.
40. **Display Operation:**
    nt. No additional input is required.
42. **Is Empty Operation:**
    nt. No additional input is required.
44. **Peek Operation:**
    nt. No additional input is required.
46. **Exit Operation:**
47. No additional input is required.

Output Format

The output will vary depending on the selected option:

48. **Push Operation:**
    nt. If the stack is not full (no overflow), the output will be: **Successfully pushed.**If memory allocation fails, it will print: **Stack is overflow.**
50. **Pop Operation:**
    nt. If the stack is not empty, it will print: **Popped value = X** where X is the value removed from the stack.If the stack is empty, it will print: **Stack is underflow.**
52. **Display Operation:**
    nt. If the stack is not empty, it will print: **Elements of the stack are : X Y Z ...** where X, Y, Z, etc., are the elements from top to bottom.If the stack is empty, it will print: **Stack is empty.**
54. **Is Empty Operation:**
    nt. If the stack is empty, it will print: **Stack is empty.**If the stack is not empty, it will print: **Stack is not empty.**
56. **Peek Operation:**
    nt. If the stack is not empty, it will print: **Peek value = X** where X is the top element of the stack.If the stack is empty, it will print: **Stack is underflow.**
58. **Exit Operation:**
59. The program terminates with no additional output.

Source Code:

StackUsingLL.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "StackOperationsLL.c"

int main() {
        int op, x;
        while(1) {
                printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d", &x);
                                push(x);
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                peek();
                                break;
                        case 6:
                                exit(0);
                }
        }
}
```

StackOperationsLL.c

```
//write your code here
struct stack{
int data;
struct stack *next;
};
typedef struct stack *STACK;
STACK top=NULL;
STACK createNode(){
        STACK newnode=(STACK)malloc(sizeof(struct stack));
        newnode->next=NULL;
        return newnode;
}
void pop()
{
        if(top==NULL)
                printf("Stack is underflow.\n");
        else
        {
                STACK temp=top;

                int tempdata=top->data;
                top=top->next;
                free(temp);
                printf("Popped value = %d\n",tempdata);
        }
}
void push(int element) {
        STACK newnode=createNode();
        newnode->data=element;
        if(top==NULL)
                top=newnode;
        else
        {
                newnode->next=top;
                top=newnode;
        }
        printf("Successfully pushed.\n");
}
void display(){
        STACK temp=top;
        if(top==NULL)
                printf("Stack is empty.\n");
        else{
                printf("Elements of the stack are : ");
                while(temp!=NULL){
                        printf("%d ",temp->data);
                        temp=temp->next;
                }
                printf("\n");
        }
}
void peek(){
        if(top==NULL)
                printf("Stack is underflow.\n");
        else{
```

```
}
void isEmpty(){
        if(top==NULL)
                printf("Stack is empty.\n");
        else
                printf("Stack is not empty.\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
33
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
22
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
55
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
66
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 66 55 22 33
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 66
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2

Popped value = 55
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Elements of the stack are : 22 33
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 22
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is not empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
3
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
23
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
1
Enter element :
24
Successfully pushed.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option :
3
Elements of the stack are : 24 23
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
5
Peek value = 24
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 24
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Popped value = 23
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
2
Stack is underflow.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
4
Stack is empty.
1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit
Enter your option :
6

S.No: 23	Exp. Name: ***C program to evaluate a Postfix expression***	Date: 2024-05-31

Aim:

C program to evaluate a postfix expression.

Write the code in the functions **isEmpty()**, **push(int x)**, **pop()** and **evaluatePostfix(char *e)** in the below program according to hints given as comment lines.

Input Format

- The user will provide a postfix expression as a single string of characters. The expression can contain digits (0-9) and operators (+, -, *, /, %).

Output Format

- If the postfix expression is valid, the program prints the result of the evaluation in the format: **Result : <result>**
- If the postfix expression is invalid (e.g., insufficient operands for the operators or extra operands remaining), the program prints: **Invalid postfix expression.** Carefully observe the print statement and add '.' at end of it

Source Code:

```
PostfixEvaluation.c
```

```
#include <ctype.h>
#include <stdio.h>
#include<stdlib.h>
#define STACK_MAX_SIZE 20
int stack[STACK_MAX_SIZE];
int top=-1;
int isoperator(char symbol){
        if(symbol=='+'||symbol=='-'||symbol=='*'||symbol=='/'){
                return 1;
        }
        return 0;
}
int isEmpty() {
        if(top==-1)
                return 1;
        else
                return 0;
}
void push(int x) {
        if(top>=STACK_MAX_SIZE){
                printf("STACK overflow\n");
                return;
        }
        top++;
        stack[top]=x;
}
int pop() {
        if(top<0) {
                printf("Invalid postfix expression.\n");
                exit(0);
        }
        int item=stack[top];
        top--;
        return item;
}
void evaluatePostfix(char * expression) {
        int i=0;
        char symbol =expression[i];
        int operand1,operand2,result;
        while(symbol!='\0'){
                if(symbol>='0' && symbol<'9'){
                        int num = symbol - '0';
                        push(num);
                }
                else if(isoperator(symbol)) {
                        operand2=pop();
                        operand1=pop();
                        switch(symbol){
                                case'+':result=operand1+operand2;
                        break;
                        case'-':result=operand1-operand2;
                        break;
                        case'*':result=operand1*operand2;
                                break;
                                case'/':result=operand1/operand2;
```

```
                        push(result);
                }
                i++;
                symbol=expression[i];
        }
        result=pop();
        if(top==-1)
                printf("Result : %d\n",result);
        else
                printf("Invalid postfix expression.\n");
}
int main() {
        char exp[220];
        char *e, x;
        printf("Enter the postfix expression : ");
        scanf("%s",exp);
        e = exp;
        evaluatePostfix(e);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the postfix expression :
234+-
Result : -5

Test Case - 2
User Output
Enter the postfix expression :
-456+5+
Invalid postfix expression.

S.No: 24	Exp. Name: ***Check for the balanced parenthesis using a stack***	Date: 2024-05-31

Aim:

Write a C program to check whether an expression consists of balanced parenthesis or not using stack

Source Code:

BalancedParenthesis.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX_SIZE 100
char stack[MAX_SIZE];
int top = -1;
void push(char data) {
        if (top == MAX_SIZE - 1) {
                return;
        }
        top++;
        stack[top] = data;
}
char pop() {
        if (top == -1) {
                return ' ';
        }
        char data = stack[top];
        top--;
        return data;
}
int is_matching_pair(char char1, char char2) {
        if(char1 =='(' && char2 ==')') {
                return 1;
        } else if (char1 == '[' && char2 ==']') {
                return 1;
        } else if (char1 == '{' && char2 =='}') {
                return 1;
        } else {
                return 0;
        }
}
int isBalanced(char* text) {
        int i;
        for (i = 0; i<strlen(text); i++) {
                if (text[i] =='(' || text[i] == '[' || text[i] == '{') {
                        push(text[i]);
                } else if (text[i] == ')' || text[i] ==']' || text[i] == '}') {
                        if (top == -1) {
                                return 0;
                        } else if (!is_matching_pair(pop(), text[i])) {
                        return 0;
                        }
                }
        }
        if(top == -1) {
                return 1;
        } else {
                return 0;
        }
}
int main() {
        char text[MAX_SIZE];
        printf("Enter an expression: ");
        scanf("%s", text);
```

```
        else
                printf("not balanced\n");
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter an expression:
1+2*3+(3+4)
balanced

Test Case - 2
User Output
Enter an expression:
1+2*(3+([4+5])
not balanced

S.No: 25	Exp. Name: ***C program to implement Operations on Queue using static Array***	Date: 2024-05-31

Aim:

Write a program to implement queue operations using static arrays

Source Code:

QueueUsingArray.c

```
#include <stdlib.h>
#include <stdio.h>
#include "QueueOperations.c"
int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
        return 0;
}
```

QueueOperations.c

```
#define maxsize 50
int front=-1,rear=-1;
int queue[maxsize];
void enqueue(int item)
{
        if(rear==maxsize-1) {
                printf("Queue is overflow.\n");
                return;
}
        if((front==-1)&&(rear==-1)){
                front=0;
                rear=0;
        }
        else
                rear=rear+1;
        queue[rear]=item;
        printf("Successfully inserted.\n");
}
void dequeue(){
        int item;
        if(front==-1||front>rear){
                printf("Queue is underflow.\n");
                return;
        }
        else{
                item=queue[front];
                if(front==rear){
                        front=-1;
                        rear=-1;
                }
                else
                                front=front+1;
                printf("Deleted element = %d\n",item);
        }
}
void display(){
        int i;
        if(rear==-1)
                printf("Queue is empty.\n");
        else{
                printf("Elements in the queue : ");
                for(i=front;i<=rear;i++)
                        printf("%d ",queue[i]);
                printf("\n");
        }
}
void isEmpty()
{
        if(front==-1)
                printf("Queue is empty.\n");
        else
                printf("Queue is not empty.\n");
}
void size()
{
```

```
                cnt=rear-front+1;
        printf("Queue size : %d\n",cnt);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
14
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
78
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
53
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 14 78 53
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit

Enter your option :
5
Queue size : 3
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
25
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 25
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
65
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
3
Elements in the queue : 65
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4
Queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
2
Deleted element = 65
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
4

Queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
1
Enter element :
63
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
5
Queue size : 1
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Enter your option :
6

S.No: 26	Exp. Name: ***Queue using Linked Lists***	Date: 2024-05-31

Aim:

Write a program that allows users to perform the following operations on a queue:

60. Enqueue an element (add to the rear).
61. Dequeue an element (remove from the front).
62. Display all elements in the queue.
63. Check if the queue is empty.
64. Get the size of the queue.
65. Exit the program.

Input Format:

The program displays a menu with the following options:

66. Enqueue
67. Dequeue
68. Display
69. Is Empty
70. Size
71. Exit

The user selects an option by entering a number corresponding to the desired operation.

For the "**Enqueue**" operation, the user is prompted to enter the integer element to be added to the queue.

Output Format:

For each operation, the program outputs the result:

72. For **Enqueue**, print: **Successfully inserted**
73. For **Dequeue**, print: **Deleted value: X** where X is the dequeued element, or **Queue underflow** if the queue is empty.
74. For **Display**, print: **Elements: A B C ...** showing all elements in the queue or **Queue is empty** if there are no elements.
75. For **Is Empty**, print: **Queue is empty** or **Queue is not empty**.
76. For **Size**, print: **Queue size: N** where N is the number of elements in the queue.
77. For **Exit**, terminate the program without additional output.

Source Code:

QueueUsingLL.c

```c
#include <stdlib.h>
#include <stdio.h>
#include "QueueOperationsLL.c"
int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit\n");
                printf("Option: ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("element: ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
}
```

QueueOperationsLL.c

```
struct node{
int data;
struct node* next;
};
typedef struct node *NODE;
NODE front=NULL,rear=NULL;
void enqueue(int item){
        NODE ptr;
        ptr=(struct node*)malloc(sizeof(struct node));
        ptr->data=item;
        printf("Successfully inserted\n");
        if(front==NULL){
                front=ptr;
                rear=ptr;
                front->next=NULL;
                rear->next=NULL;
        }
        else{
                rear->next=ptr;
                rear=ptr;
                rear->next=NULL;
        }
}
void dequeue(){
        NODE ptr;
        if(front==NULL)
                printf("Queue is underflow\n");
        else
        {
                ptr=front;
                printf("Deleted value: %d\n",ptr->data);
                front=front->next;
                free(ptr);
        }
}
void display()
{
        NODE ptr;
        ptr=front;
        if(front==NULL)
                printf("Queue is empty\n");
        else{
                printf("Elements: ");
                while(ptr!=NULL){
                        printf("%d ",ptr->data);
                        ptr=ptr->next;
                }
                printf("\n");
        }
}
void isEmpty()
{
if(front==NULL)
        printf("Queue is empty\n");
        else
```

```
void size()
{
        int cnt=0;
        NODE ptr=front;
        while(ptr!=NULL){
                cnt++;
                ptr=ptr->next;
        }
        printf("Queue size: %d\n",cnt);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Queue is underflow
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
3
Queue is empty
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
4
Queue is empty
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
5
Queue size: 0
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
44
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
55
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
66

Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
67
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
3
Elements: 44 55 66 67
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Deleted value: 44
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Deleted value: 55
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
5
Queue size: 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
4
Queue is not empty
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
23
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
234
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1

element:
45
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
1
element:
456
Successfully inserted
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Deleted value: 23
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
3
Elements: 234 45 456
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
2
Deleted value: 234
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
3
Elements: 45 456
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
4
Queue is not empty
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
5
Queue size: 2
1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit
Option:
6

S.No: 27	Exp. Name: ***Simulation of a simple printer queue system.***	Date: 2024-05-31

Aim:
Develop a C program to simulate a simple printer queue system.

Note: Before exiting the printer system, all the jobs must be done.
Source Code:

PrinterQueue.c

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
typedef struct node *NODE;
NODE front=NULL, rear=NULL;

void enqueue(int item)
{
        NODE ptr;
        ptr = (struct node *)malloc(sizeof(struct node));
        ptr->data=item;
        printf("Job %d added\n",item);
        if(front==NULL) {
                front=ptr;
                rear=ptr;
                front->next=NULL;
                rear->next=NULL;
        }
        else{
                rear->next=ptr;
                rear=ptr;
                rear->next= NULL;
        }
}
void dequeue()
{
        NODE ptr;
        if(front==NULL)
                printf("No job to dequeue\n");
        else
        {
                ptr=front;
                printf("Job %d removed from queue and sent to the printer\n",ptr->data);
                front=front->next;
                free(ptr);
        }
}
int main() {
        int op,x;
        while(1) {
                printf("Printer Queue System\n");
                printf("1. Add a job to queue\n");
                printf("2. Process the next job\n");
                printf("3. Exit\n");
                printf("Choice: ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                        printf("Job ID: ");
                        scanf("%d",&x);
                        enqueue(x);
```

```
                        dequeue();
                        break;
                        case 3:
                        printf("Exiting\n");
                        while(1)  {
                                if(front==NULL)
                                        exit(0);
                                else
                                        dequeue();
                        }
                        exit(0);
                        default:
                        printf("Invalid choice\n");
                }
        }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
2
No job to dequeue
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
1
Job ID:
1245
Job 1245 added
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
1
Job ID:
2345
Job 2345 added
Printer Queue System
1. Add a job to queue
2. Process the next job

3. Exit
Choice:
1
Job ID:
2145
Job 2145 added
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
5
Invalid choice
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
2
Job 1245 removed from queue and sent to the printer
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
2
Job 2345 removed from queue and sent to the printer
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
2
Job 2145 removed from queue and sent to the printer
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
2
No job to dequeue
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
3
Exiting

Test Case - 2
User Output
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
1
Job ID:
1245
Job 1245 added
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
1
Job ID:
3654
Job 3654 added
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
1
Job ID:
8569
Job 8569 added
Printer Queue System
1. Add a job to queue
2. Process the next job
3. Exit
Choice:
3
Exiting
Job 1245 removed from queue and sent to the printer
Job 3654 removed from queue and sent to the printer
Job 8569 removed from queue and sent to the printer

S.No: 28	Exp. Name: ***Circular Queues using arrays***	Date: 2024-05-31

Aim:

Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX = 6)

a) Insert an Element on to Circular QUEUE

b) Delete an Element from Circular QUEUE

c) Demonstrate Overflow and Underflow situations on Circular QUEUE

d) Display the status of Circular QUEUE

e) Exit

Support the program with appropriate functions for each of the above operations.

Source Code:

cQue.c

```
#include<stdio.h>
#include<stdlib.h>
#define max 6
int queue[max];
int front=-1;
int rear=-1;
void enqueue(int element)
{
        if(front==-1 && rear==-1)
        {
                front=0;
                rear=0;
                queue[rear]=element;
        }
        else if((rear+1)%max==front)
        {
                printf("~~~Circular Queue Overflow~~~\n");
        }
        else{
                rear=(rear+1)%max;
                queue[rear]=element;
        }
}
void dequeue()
{
        if((front==-1) && (rear==-1))
                printf("~~~Circular Queue Underflow~~~\n");
        else if(front==rear)
        {
                printf("Deleted element from the queue is: %d\n",queue[front]);
                front=-1;
                rear=-1;
        }
        else{
                printf("Deleted element from the queue is: %d\n",queue[front]);
                front=(front+1)%max;
        }
}
void display()
{
        int i=front;
        if(front==-1 && rear==-1)
                printf("~~~Circular Queue Empty~~~\n");
        else
        {
                printf("Circular Queue contents are:\n");
                while(i!=rear)
                        {
                                printf("%d ",queue[i]);
                                i=(i+1)%max;
                        }
                printf("%d\n",queue[i]);
        }
}
int main()
```

```
        while(1){
                printf("~~Main Menu~~\n");
                printf("=> 1. Insertion and Overflow Demo\n");
                printf("=> 2. Deletion and Underflow Demo\n");
                printf("=> 3. Display\n");
                printf("=> 4. Exit\n");
                printf("Enter Your Choice: ");
                scanf("%d",&choice);
                switch(choice) {
                        case 1:
                        printf("Enter the element to be inserted: ");
                        scanf("%d",&x);
                        enqueue(x);
                        break;
                        case 2:
                        dequeue();
                        break;
                        case 3:
                        display();
                        break;
                        case 4: exit(0);
                        default:
                        printf("Please enter a valid choice\n");
                }
        }
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
1
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
2

```
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
3
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
4
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
5
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
3
Circular Queue contents are:
1 2 3 4 5
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
6
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
```

```
Enter Your Choice:
3
Circular Queue contents are:
1 2 3 4 5 6
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
7
~~~Circular Queue Overflow~~~
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
3
Circular Queue contents are:
1 2 3 4 5 6
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
2
Deleted element from the queue is: 1
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
2
Deleted element from the queue is: 2
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
2
Deleted element from the queue is: 3
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
```

=> 4. Exit
Enter Your Choice:
3
Circular Queue contents are:
4 5 6
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
4

Test Case - 2
User Output
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
3
~~Circular Queue Empty~~~
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
3
~~Circular Queue Empty~~~
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
2
~~Circular Queue Underflow~~~
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
1
Enter the element to be inserted:
4
~~Main Menu~~

=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
4

Test Case - 3
User Output
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
5
Please enter a valid choice
~~Main Menu~~
=> 1. Insertion and Overflow Demo
=> 2. Deletion and Underflow Demo
=> 3. Display
=> 4. Exit
Enter Your Choice:
4

S.No: 29	Exp. Name: ***Implementation of Circular Queue using Linked List***	Date: 2024-05-31

Aim:

Write a program to implement `circular queue` using **linked lists**.

Source Code:

CQueueLL.c

```
#include <stdlib.h>
#include <stdio.h>
#include "CQueueOperationsLL.c"
int main() {
        int op, x;
        while(1) {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op) {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
}
```

CQueueOperationsLL.c

ID: 23K61A4752
2023-2027-CIC
Sasi Institute of Technology and Engineering (Autonomous)

```
struct queue {
        int data;
        struct queue *next;
};
typedef struct queue *CircularQueue;
CircularQueue front = NULL, rear = NULL;
//complete the below dequeue() and enqueue() functions
void dequeue() {
        CircularQueue temp =front;
        if((front==NULL)&&(rear==NULL))
                printf("Circular queue is underflow.\n");
        else if(front==rear){
                front=rear=NULL;
                printf("Deleted value = %d\n",temp->data);
                free(temp);
        }
        else{
                front=front->next;
                rear->next=front;
                printf("Deleted value = %d\n",temp->data);
                free(temp);
        }


}

void size() {
        int count =0;
        if(front == NULL) {
                printf("Circular queue size : 0\n");
                return;
        }
        CircularQueue temp = front;
        do {
                temp = temp -> next;
                count = count + 1;
        } while(temp != front);
        printf("Circular queue size : %d\n",count);
}

void isEmpty() {
        if(front == NULL ) {
                printf("Circular queue is empty.\n");
        } else {
                printf("Circular queue is not empty.\n");
        }
}

void enqueue(int element) {
        CircularQueue newnode;
        newnode=(CircularQueue)malloc(sizeof(struct queue));
        newnode->data=element;
        newnode->next=NULL;
        if((rear==NULL)&&(front==NULL)) {
                front=rear=newnode;
```

```
        else{
                rear->next=newnode;
                rear=newnode;
                newnode->next=front;
        }
        printf("Successfully inserted.\n");




}

void display() {
        if(front == NULL) {
                printf("Circular queue is empty.\n");
        } else {
                CircularQueue temp = front;
                printf("Elements in the circular queue : ");
                do {
                        printf("%d ", temp -> data);
                        temp = temp -> next;
                } while(temp != front);
                printf("\n");
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
15
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
16
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
17
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :

3
Elements in the circular queue : 15 16 17
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 3
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 15
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 16
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 17
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
4
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
6

Test Case - 2
User Output
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Circular queue is underflow.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 0
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
4
Circular queue is empty.

```
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Circular queue is empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
143
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
153
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
163
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
1
Enter element :
173
Successfully inserted.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
3
Elements in the circular queue : 143 153 163 173
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 143
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
2
Deleted value = 153
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
5
Circular queue size : 2
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
4
Circular queue is not empty.
1.Enqueue 2.Dequeue 3.Display 4.Is empty 5.Size 6.Exit
Enter your option :
```

S.No: 30	Exp. Name: ***Convert an Infix Expression to Postfix and Evaluate it.***	Date: 2024-05-31

Aim:

Write a C program that uses a stack to evaluate an infix expression and convert it to a postfix expression.

Note: Only single-digit positive integers and **+, -, *, /, %** operators are allowed

Input Format

- A string representing the infix mathematical expression

Output Format

- The first line of output represents the converted postfix notation of the infix expression.
- The second line of output represents the result of evaluating the postfix expression.

Source Code:

```
EvaluateConverted.c
```

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
int numbers[50],tn=-1,to=-1;
char op[50];
void push_num(int n)
{
        numbers[++tn]=n;}
void push_op(char ch)
{
        op[++to]=ch;
}
int pop_num()
{
        return numbers[tn--];
}
int peek()
{
        return op[to];
}
char pop_op()
{
        return op[to--];
}
int infix_eval(int numbers[50],char op[50])
{
        int x,y;
        char ope;
        x=pop_num();
        y=pop_num();
        ope=pop_op();
        switch(ope)
                {
                        case '+':
                        return x+y;
                        case '-':
                        return y-x;
                        case '%':
                        return y%x;
                        case '*':
                        return x*y;
                        case '/':
                        if(x==0) {
                                printf("\nCan not divide by 0");
                                exit(0);
                        }
                        else
                                return y/x;
                }
        return 0;
}
int is_operator(char ch)
{
        return(ch=='+'||ch=='-'||ch=='*'||ch=='/'||ch=='%');
```

```
{
        switch(c)
                {
                        case '+':
                        case '-': return 1;
                        case '*':
                        case '/':
                        case '%': return 2;
                }
        return -1;
}
int eval(char exp[20])
{
        int i,num,output,r;
        char c;
        for(i=0;exp[i]!='\0';i++)
                {
                        c = exp[i];
                        if(isdigit(c)!=0)
                        {
                        num=0;
                        while (isdigit(c)) {
                                num=num*10+(c-'0');
                                i++;
                                if(i<strlen(exp))
                                        c=exp[i];
                                else
                                        break;
                        }
                        i--;
                        push_num(num);
                }
        else if(c=='(') {
                push_op(c);
        }
                        else if(c==')') {
                                while(op[to]!='(') {
                                        r=infix_eval(numbers,op);
                                        push_num(r);
                                }
                                pop_op();
                        }
                        else if(is_operator(c))
                        {
                                while(to!=-1 && precedence(c)<=precedence(op[to]))
                                        {
                                                output=infix_eval(numbers,op);
                                                push_num(output);
                                        }
                                push_op(c);
                        }
                        }
                        while(to!=-1)
                                {
                                        output=infix_eval(numbers,op);
```

```
                        return pop_num();
}
        void convertInfixToPostfix(char* expression)
{
        int i, j;
        for(i=0,j= -1; expression[i];++i)
                {
                        if(isdigit(expression[i]))
                                expression[++j]=expression[i];
                        else if(expression[i]=='(')
                                push_op(expression[i]);
                                        else if(expression[i]==')')
                                        {
                                                while(to!=-1&&peek()!='(')
                                                        expression[++j]=pop_op();
                                                if(to!=-1&&peek()!='(')
                                                        return;
                                                else
                                                        pop_op();
                                        }
                        else
                                        {
                                while(to!=-1&&precedence(expression[i])
<=precedence(peek()))
                                        expression[++j]=pop_op();
                                push_op(expression[i]);
                        }
                }
        while(to!=-1)
                expression[++j]=pop_op();
                                        expression[++j]='\0';
                                printf("Postfix expression: %s\n",expression);
}
int main()
{
        char exp[50];
        int result;
        printf("Infix expression: ");
        scanf("%s",exp);
        result=eval(exp);
        convertInfixToPostfix(exp);
        printf("Result: %d\n",result);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Infix expression:
2+3*4
Postfix expression: 234*+
Result: 14

Test Case - 2
User Output
`Infix expression:`
8%3+6*(2-1)
`Postfix expression: 83%621-*+`
`Result: 8`

S.No: 31	Exp. Name: ***Check whether the given string is palindrome or not using stack*.**	**Date: 2024-05-31**

Aim:

Create a C program to determine whether a given string is a palindrome or not using stack.

Source Code:

StringPalinUsingStack.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

struct Stack {
    int top;
    char items[100];
};
struct Stack *stackPtr;
void initStack(char *inputstring);
void push(char);
char pop();
void initStack(char *inputString)
{
        stackPtr=(struct Stack*) malloc(sizeof(struct Stack));
        stackPtr->top=-1;
        for(int i=0;inputString[i]!='\0';i++)
                push(inputString[i]);
}
void push(char ch)
{
        stackPtr->items[++stackPtr->top]=toupper(ch);
}
char pop()
{
        return stackPtr->items[stackPtr->top--];
}
int isPalindrome(char *inputString)
{
int front=0;
        initStack(inputString);
        for(int i=0;i<(strlen(inputString) / 2);i++) {
                if(stackPtr->items[stackPtr->top]==stackPtr->items[front])  {
                        pop();
                        front++;
                }
                else
                        return 0;
        }
        return 1;
}

int main() {
    char inputString[100];

    printf("String: ");
    scanf("%s", inputString);

    if (isPalindrome(inputString)) {
        printf("%s is a palindrome\n", inputString);
    } else {
        printf("%s is not a palindrome\n", inputString);
    }
```

```
    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
String:
Madam
Madam is a palindrome

Test Case - 2
User Output
String:
Aplha
Aplha is not a palindrome

S.No: 32	Exp. Name: ***Check for Symmetry of a String using Stack***	Date: 2024-05-31

Aim:
Implement a stack using C to perform comparison and check for symmetry of a String.

Note: Convert all the characters of the string to lowercase for case-insensitivity before checking for symmetry.
Source Code:

StringSymmetry.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
struct Stack {
int top;
char items[100];
};
struct Stack *stackPtr;
void initStack(char *inputString);
int isSymmetric(char *inputString);
void push(char);
char pop();
void initStack(char *inputString)
{
        stackPtr=(struct Stack*)malloc(sizeof(struct Stack));
        stackPtr->top=-1;
        for(int i=0;inputString[i]!='\0';i++)
                push(inputString[i]);
}
void push(char ch)
{
        stackPtr->items[++stackPtr->top]=toupper(ch);
}
char pop()
{
        return stackPtr->items[stackPtr->top--];
}
int isSymmetric(char *inputString)
{
        initStack(inputString);
        for(int i=0;i<(strlen(inputString) / 2);i++) {
                if(pop()!=stackPtr->items[i])
                        return 0;
        }
        return 1;
}
int main()
{
        char inputString[100];
        printf("String: ");
        scanf("%s", inputString);
        if(isSymmetric(inputString)) {
                printf("%s is symmetric\n", inputString);
        } else {
                printf("%s is not symmetric\n", inputString);
        }
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output
String:
Emualator
Emualator is not symmetric

Test Case - 2
User Output
String:
Madam
Madam is symmetric

S.No: 33	Exp. Name: ***Check for Symmetry of a String using Queue***	Date: 2024-06-06

Aim:

Implement a queue using C to perform comparison and check for symmetry of a String.

Note: Convert all the characters of the string to lowercase for case-insensitivity before checking for symmetry.

Source Code:

StringSymmetryUsingQueue.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
struct queue {
int front;
int rear;
char items[100];
};
struct queue *queuePtr;
void initQueue(char *inputString);
int IsSymmetric(char *inputString);
void Enqueue(char);
char Dequeue();
void initQueue(char *inputString) {
        queuePtr = (struct queue*)malloc(sizeof(struct queue));
        queuePtr->front = -1;
        queuePtr->rear = -1;
        for(int i = 0;inputString[i] != '\0'; i++)
                Enqueue(inputString[i]);
}
void Enqueue(char ch) {
        if(queuePtr->front == -1)
                ++queuePtr->front;
        queuePtr->items[++queuePtr->rear] = toupper(ch);
}
char Dequeue() {
        return queuePtr->items[queuePtr->front++];
}
int IsSymmetric(char *inputString)
{
        int r = strlen(inputString) - 1;
initQueue(inputString);
for(int i = 0; i < strlen(inputString) / 2; i++) {
        if(Dequeue() == queuePtr->items[r])
                r--;
        else
                return 0;
}
return 1;
}
int main() {
char inputString[100];
printf("String: ");
        scanf("%s", inputString);
if(IsSymmetric(inputString)) {
        printf("%s is symmetric\n", inputString);
}
else{
        printf("%s is not symmetric\n", inputString);
}
return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
String:
abcdcba
abcdcba is symmetric

Test Case - 2
User Output
String:
Alpha
Alpha is not symmetric

S.No: 34	Exp. Name: ***Program to insert into BST and traversal using In-order, Pre-order and Post-order***	Date: 2024-06-01

Aim:

Write a program to create a binary search tree of integers and perform the following operations using linked list.

78. Insert a node
79. In-order traversal
80. Pre-order traversal
81. Post-order traversal

Note: Write the code in **InsertAndTraversals.c** file.

Source Code:

BinarySearchTree.c

```
#include<stdio.h>
#include<stdlib.h>
#include "InsertAndTraversals.c"

void main() {
        int x, op;
        BSTNODE root = NULL;
        while(1) {
                printf("1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder
Traversal 5.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1: printf("Enter an element to be inserted : ");
                                        scanf("%d", &x);
                                        root = insertNodeInBST(root,x);
                                        break;
                        case 2:
                                        if(root == NULL) {
                                                printf("Binary Search Tree is empty.\n");
                                        }
                                        else {
                                                printf("Elements of the BST (in-order
traversal): ");
                                                inorderInBST(root);
                                                printf("\n");
                                                }
                                        break;
                        case 3:
                                        if(root == NULL) {
                                                printf("Binary Search Tree is empty.\n");
                                        }
                                        else {
                                                printf("Elements of the BST (pre-order
traversal): ");
                                                preorderInBST(root);
                                                printf("\n");
                                                }
                                        break;
                        case 4:
                                        if(root == NULL) {
                                                printf("Binary Search Tree is empty.\n");
                                        }
                                        else {
                                                printf("Elements of the BST (post-order
traversal): ");
                                                postorderInBST(root);
                                                printf("\n");
                                                }
                                        break;
                        case 5:
                                        exit(0);
                }
        }
}
```

InsertAndTraversals.c

```
struct node {
        int data;
        struct node *left, *right;
};

typedef struct node *BSTNODE;

BSTNODE newNodeInBST(int item) {
        BSTNODE temp =  (BSTNODE)malloc(sizeof(struct node));
        temp->data = item;
        temp->left = temp->right = NULL;
        return temp;
}

BSTNODE insertNodeInBST(BSTNODE node, int ele) {
        if(node == NULL)
        {
                printf("Successfully inserted.\n");
                return newNodeInBST(ele);
        }
        if(ele < node->data)
                node->left = insertNodeInBST(node->left, ele);
        else if(ele > node->data)
                node->right = insertNodeInBST(node->right, ele);
        else
                printf("Element already exists in BST.\n");
        return node;
}

void inorderInBST(BSTNODE root) {
        if(root != NULL) {
                inorderInBST(root -> left);
                printf("%d ", root->data);
                inorderInBST(root->right);
        }
}

void preorderInBST(BSTNODE root) {
if(root != NULL){
        printf("%d ", root->data);
        preorderInBST(root->left);
        preorderInBST(root->right);
}
}

void postorderInBST(BSTNODE root) {
    if(root != NULL) {
                postorderInBST(root->left);
                postorderInBST(root->right);
                printf("%d ",root->data);
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
54
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
28
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
62
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
2
Elements of the BST (in-order traversal): 28 54 62
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
3
Elements of the BST (pre-order traversal): 54 28 62
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
4
Elements of the BST (post-order traversal): 28 62 54
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
5

Test Case - 2
User Output
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
100
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit

```
Enter your option :
1
Enter an element to be inserted :
20
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
200
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
10
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
30
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
150
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
300
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
2
Elements of the BST (in-order traversal): 10 20 30 100 150 200 300
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
3
Elements of the BST (pre-order traversal): 100 20 10 30 200 150 300
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
4
Elements of the BST (post-order traversal): 10 30 20 150 300 200 100
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
```

5

Test Case - 3
User Output
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
12
Successfully inserted.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
1
Enter an element to be inserted :
12
Element already exists in BST.
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
2
Elements of the BST (in-order traversal): 12
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
3
Elements of the BST (pre-order traversal): 12
1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit
Enter your option :
5

S.No: 35	Exp. Name: ***Binary Search Tree using Linked List***	Date: 2024-06-01

Aim:

Write a program to create a binary search tree of integers and perform the following operations using linked list.

82. Insert a node.
83. Delete a node.

Source Code:

BinarySearchTree.c

```c
#include <stdio.h>
#include <stdlib.h>
struct node {
  int key;
  struct node *left, *right;
};

// Creation
struct node *newNode(int item) {
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->key = item;
  temp->left = temp->right = NULL;
  return temp;
}

// Inorder Traversal using recursion
void inorder(struct node *root) {
  if (root != NULL) {
    // Traverse left
    inorder(root->left);
    // Traverse root
    printf("%d ", root->key);
    // Traverse right
    inorder(root->right);
  }
}

void preorder(struct node *root) {
  if (root != NULL) {
    // Traverse left
     printf("%d ", root->key);
    preorder(root->left);
    // Traverse right
    preorder(root->right);
  }
}
// Find the inorder successor
struct node *minValueNode(struct node *node) {
  struct node *current = node;
  // Find the leftmost leaf
  while (current && current->left != NULL)
    current = current->left;
  return current;
}

// Insert a node in BST
struct node *insert(struct node *node, int key) {
  // write your code here to perform insertion operation
if(node == NULL)
        return newNode(key);
        if(key < node->key)
                node->left = insert(node->left, key);
        else if(key > node->key)
                node->right = insert(node->right, key);
        return node;
```

```
}

// Deleting a node
struct node *deleteNode(struct node *root, int key) {

  // write your code here to perform deletion operation
if(root == NULL)
        return root;
        if(key < root->key)
                root->left = deleteNode(root->left, key);
        else if(key > root->key)
                root->right = deleteNode(root->right, key);
                else {
                        if(root->left == NULL) {
                                struct node *temp = root->right;
                                free(root);
                                return temp;
                        }
                        else if(root->right == NULL) {
                                struct node *temp = root->left;
                                free(root);
                                return temp;
                        }
                        struct node *temp = minValueNode(root->right);
                        root->key = temp->key;
                        root->right = deleteNode(root->right, temp->key);
                }
        return root;



}

// Driver code
int main() {
  struct node *root = NULL;
  int n,data;
  printf("Enter how many nodes you want to insert in BST :");
  scanf("%d",&n);
  for( int i =0 ; i < n ; i++){
      printf("Enter the value: ");
      scanf("%d", &data);
      root = insert(root, data);
  }

  printf("Inorder traversal(Always gives ascending order): ");
  inorder(root);
  printf("\nPreorder traversal: ");
  preorder(root);
  printf("\nEnter the data to delete: ");
  scanf("%d",&data);

  printf("After deleting %d\n",data);
  root = deleteNode(root, data);
  printf("Inorder traversal: ");
```

```
  preorder(root);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter how many nodes you want to insert in BST :
5
Enter the value:
10
Enter the value:
9
Enter the value:
20
Enter the value:
45
Enter the value:
8
Inorder traversal(Always gives ascending order): 8 9 10 20 45
Preorder traversal: 10 9 8 20 45
Enter the data to delete:
8
After deleting 8
Inorder traversal: 9 10 20 45
Preorder traversal: 10 9 20 45

Test Case - 2
User Output
Enter how many nodes you want to insert in BST :
6
Enter the value:
45
Enter the value:
15
Enter the value:
65
Enter the value:
25
Enter the value:
35
Enter the value:
95
Inorder traversal(Always gives ascending order): 15 25 35 45 65 95
Preorder traversal: 45 15 25 35 65 95

Enter the data to delete:
15
After deleting 15
Inorder traversal: 25 35 45 65 95
Preorder traversal: 45 25 35 65 95

S.No: 36	Exp. Name: ***Collision resolution techniques : linear probing***	Date: 2024-06-01

Aim:

Write a C program to implement Linear Probing.

Source Code:

HashingMain2.c

```
#include <stdio.h>
#include <stdlib.h>
#include "HashingLinearProbing.c"
int main() {
        int x, op, i = 0;
        for (i = 0; i < SIZE; i++)
                HashTable[i] = -1;
        while (1) {
                printf("1.Insert 2.Delete 3.Search 4.Print 5.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch (op) {
                        case 1: printf("Enter an element to be inserted : ");
                                        scanf("%d", &x);
                                        insert(x);
                                        break;
                        case 2:
                                        printf("Enter an element to be deleted : ");
                                        scanf("%d", &x);
                                        deleteElement(x);
                                        break;
                        case 3:
                                        printf("Enter an element to be searched : ");
                                        scanf("%d", &x);
                                        search(x);
                                        break;
                        case 4:
                                        print();
                                        break;
                        case 5: exit(0);
                }
        }
}
```

HashingLinearProbing.c

```
#define SIZE 10
int HashTable[SIZE];
int hash(int x) {
        return x % SIZE;
}
void insert(int x) {
        int index = hash(x);
        int start = index;
        while(HashTable[index] != -1) {
                index = (index + 1) % SIZE;
                if(index == start) {
                        printf("Hash table is full.So cannot insert the element.\n");
                        return;
                }
        }
        HashTable[index] = x;
        printf("Successfully inserted.\n");
}
void deleteElement(int x) {
        int index = hash(x);
        int start = index;
        while(HashTable[index] != x) {
                index = (index + 1) % SIZE;
                if(index == start) {
                        printf("Element not found.So cannot delete the element.\n");
                        return;
                }
        }
        HashTable[index] = -1;
        printf("Successfully deleted.\n");
}
void search(int x) {
        int index = hash(x);
        int start = index;
        while(HashTable[index] != x) {
                index = (index + 1) % SIZE;
                if(index == start) {
printf("Element not found.\n");
                return;
                }
        }
        printf("Element found.\n");
}
void print() {
        int i;
        for(i = 0; i < SIZE; i++) {
                if(HashTable[i] != -1)
                        printf("[%d]=>%d ", i, HashTable[i]);
        }
        printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
11
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
22
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
33
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
43
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
53
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[1]=>11 [2]=>22 [3]=>33 [4]=>43 [5]=>53
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
44
Successfully inserted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[1]=>11 [2]=>22 [3]=>33 [4]=>43 [5]=>53 [6]=>44
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :

3
Enter an element to be searched :
34
Element not found.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
2
Enter an element to be deleted :
33
Successfully deleted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[1]=>11 [2]=>22 [4]=>43 [5]=>53 [6]=>44
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
5

S.No: 37	Exp. Name: ***Collision resolution techniques - Separate chaining***	Date: 2024-06-01

Aim:

Write a C program to implement Separate Chaining.

Source Code:

HashingMain4.c

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include "HashingSeperateChaining.c"
int main() {
        int x, op, i=0;
        for(i=0;i<SIZE;i++)
                HashTable[i]=NULL;
        while(1) {
                printf("1.Insert 2.Delete 3.Search 4.Print 5.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op) {
                        case 1: printf("Enter an element to be inserted : ");
                                        scanf("%d", &x);
                                        insert(x);
                                        break;
                        case 2:
                                        printf("Enter an element to be deleted : ");
                                        scanf("%d", &x);
                                        deleteElement(x);
                                        break;
                        case 3:
                                        printf("Enter an element to be searched : ");
                                        scanf("%d", &x);
                                        search(x);
                                        break;
                        case 4:
                                        print();
                                        break;
                        case 5: exit(0);
                }
        }
}
```

HashingSeperateChaining.c

```
#define SIZE 10

struct node {
        int data;
        struct node * next;
};
struct node * HashTable[SIZE];

int hash(int x) {
        return x % SIZE;
}

struct node * newNode(int x) {
        struct node * temp = (struct node *) malloc(sizeof(struct node *));
        temp->data = x;
        temp->next = NULL;
        return temp;
}
void insert(int x) {
        int key = x % SIZE;
        if(HashTable[key] == NULL)
                HashTable[key] = newNode(x);
        else
        {
                struct node *temp = HashTable[key];
                HashTable[key]=newNode(x);
                HashTable[key]->next =temp;
        }
}
void deleteElement(int x) {
        int key = x % SIZE;
        struct node *temp = HashTable[key], *dealloc;
        if(temp != NULL)
        {
                if(temp->data == x)
                {
                        dealloc = temp;
                        HashTable[key]=temp->next;
                        free(dealloc);
                        printf("Successfully deleted.\n");
                        return;
                }
                else
                {
                        while(temp->next)
                                {
                                        if(temp->next->data == x)
                                        {
                                                dealloc = temp->next;
                                                temp->next = temp->next->next;
                                                free(dealloc);
                                                printf("Successfully deleted.\n");
                                                return;
                                        }
                                        temp = temp->next;
```

```
        }
        printf("Element not found. So cannot delete.\n");
}

void search(int x) {
        int key = x % SIZE;
        struct node *temp = HashTable[key];
        while(temp)
                {
                        if(temp->data == x) {
                                printf("Element found.\n");
                                return;
                        }
                        temp = temp->next;
                }
        printf("Element not found.\n");
}
void print() {
        int i,flag=0;
        for(i = 0; i< SIZE; i++)
                {
                        struct node *temp = HashTable[i];
                        if (temp!=NULL) printf("[%d]=> ",i);
                        while(temp)
                                {
                                        printf("%d ",temp->data);
                                        temp = temp->next;
                                        flag=1;
                                }
                }
        if(flag==0)
                printf("Empty HashTable\n");
        else
                printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
113
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
134
1.Insert 2.Delete 3.Search 4.Print 5.Exit

Enter your option :
1
Enter an element to be inserted :
56
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
1
Enter an element to be inserted :
76
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[3]=> 113 [4]=> 134 [6]=> 76 56
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
2
Enter an element to be deleted :
76
Successfully deleted.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
3
Enter an element to be searched :
44
Element not found.
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
4
[3]=> 113 [4]=> 134 [6]=> 56
1.Insert 2.Delete 3.Search 4.Print 5.Exit
Enter your option :
5

S.No: 38	Exp. Name: ***Implement a simple cache using hashing.***	Date: 2024-06-01

Aim:

Implement a C program to create a simple cache using hashing. The program must input key-value pairs and retrieve values based on user input. The program should handle collisions using linear probing and display appropriate messages for full cache or key not found scenarios. Provide a menu-driven interface to facilitate user interaction.

Source Code:

Cache.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define SIZE 10
struct HashTable {
int keys[SIZE];
char values[SIZE][10];
int occupied[SIZE];
};
void initialise(struct HashTable *table) {
        for(int i = 0; i< SIZE; i++)  {
                table->keys[i] = -1;
                table->occupied[i] = 0;
        }
}
int hash(int key) {
        return key % SIZE;
}
void insert(struct HashTable *table, int key, char *value) {
        int index = hash(key);
        while(table->occupied[index]) {
                index = (index + 1) % SIZE;
        }
        table->keys[index] = key;
        strcpy(table->values[index], value);
        table->occupied[index] = 1;
}
void search(struct HashTable *table, int key) {
        int index = hash(key);
        while(table->occupied[index]) {
                if(table->keys[index] == key) {
printf("Value for key %d: %s\n", key, table->values[index]);
                return;
                }
                index = (index + 1) % SIZE;
        }
        printf("Key %d not found in the cache\n", key);
}
int main() {
        struct HashTable table;
        int op, x, key;
        char value[10];
        initialise(&table);
        while(1) {
                printf("Cache Menu:\n");
                printf("1. Insert a key-value pair\n");
                printf("2. Retrieve value for a key\n");
                printf("3. Exit\n");
                printf("Choice: ");
                scanf("%d", &op);
                switch(op) {
                        case 1:
                        printf("Key: ");
                        scanf("%d", &key);
                        printf("Value: ");
```

```
                        break;
                        case 2:
                        printf("Key for retrieval: ");
                        scanf("%d", &key);
                        search(&table, key);
                        break;
                        case 3:
                        printf("Exiting\n");
                        exit(0);
                        default:
                        printf("Invalid choice\n");
                }
        }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
2
Key for retrieval:
10
Key 10 not found in the cache
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
1
Key:
10
Value:
010
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
1
Key:
20
Value:
020
Cache Menu:

1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
2
Key for retrieval:
30
Key 30 not found in the cache
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
303
Invalid choice
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
2
Key for retrieval:
30
Key 30 not found in the cache
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
10
Invalid choice
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
10
Invalid choice
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
2
Key for retrieval:
10
Value for key 10: 010
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key

3. Exit
Choice:
2
Key for retrieval:
20
Value for key 20: 020
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
2
Key for retrieval:
10
Value for key 10: 010
Cache Menu:
1. Insert a key-value pair
2. Retrieve value for a key
3. Exit
Choice:
3
Exiting